Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Komprese DNA

DNA Compression

2014 **Bc. Michal Sionkala**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Michal Sionkala**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Komprese DNA
DNA Compression

Zásady pro vypracování:

Cílem práce je navrhnout metody pro kompresi DNA řetězců uložených v standardních souborech, při dosažení maximální efektivity výsledné komprese.

Práce bude obsahovat:
1. Analýzu stavu komprese DNA.
2. Návrh vlastní metody pro kompresi.
3. Implementace algoritmu.
4. Testování metody a porovnání s existujícími algoritmy.

Seznam doporučené odborné literatury:

[1] Data Compression: The Complete Reference, David Salomon, 4. ed., Springer, 2007

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2014

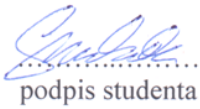
doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *25. července 2014*



podpis studenta

Poděkování

Rád bych poděkoval panu doc. Ing. Janu Platošovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této diplomové práce.

Abstrakt

Komprese DNA sekvencí je považována za obtížný úkol. Její význam nehraje roli pouze pro úsporu diskového prostoru a využití sítě při přenosu souborů s genomy. Přínosem je také rozpoznávání modelů uvnitř biologických sekvencí a určování evoluční vzdálenosti mezi organizmy. Tato diplomová práce začíná náhledem do biologie a základním poznáním struktury DNA sekvencí. Následuje chronologicky seřazený rozbor vybraných kompresních programů. Je popsána jejich strategie, algoritmy a rozdílné způsoby řešení společných dílčích problémů. Na základě získaných znalostí a poznatků z testování je navržena vlastní metoda, která je implementována v programu DNAcod. Dosažené výsledky jsou porovnány s ostatními kompresními nástroji.

Klíčová slova

DNA; biologická sekvence; komprese; kódování; algoritmus; bioinformatika;

Abstract

DNA compression is considered as a challenging task. It is not useful just for saving the disk space and network bandwidth while transferring genome file. The benefit is also in recognition of the patterns in biological sequences and measuring the evolutionary distance between organisms. This thesis starts with insight into Biology and basic knowledge of DNA sequence structure. It is followed by chronologically ordered chapters with analysis of chosen compression programs. Their strategies, algorithms and different types of solution for common partial issues are described. Based on the gained knowledge and experience from testing own compression method has been designed and then implemented in DNAcod program. Achieved outcome numbers are compared with other compression tools results.

Key words

DNA; biological sequence; compression; coding; algorithm; bioinformatics;

Seznam použitých zkratek

Zkratka	Význam
ASCII	American Standard Code for Information Interchange
báze A	Adenin
báze C	Cytosin
báze G	Guanin
báze T	Tymin
bp	base pair
bpb	bits per base
BWT	Burrows-Wheeler Transform
CTW	Context Tree Weighting
DNA	Deoxyribonucleic Acid
EOF	End of file
KB	KiloByte
LZ	Lempel-Ziv
NML	Normalized Maximum Likelihood
PPM	Prediction by Partial Matching

Obsah

Úvod

DNA je nosičem genetické informace takřka všech autonomních forem života. Umožňuje přenos dědičných znaků na další generaci. Má nesmírně důležitou schopnost tvořit svoji kopii – replikace. Předurčuje vývoj a vlastnosti celého organizmu. Programuje život.

Vědci se neustále snaží rozluštit její kód a lépe porozumět jejím částem. Zkoumají význam jednotlivých genů a porovnávají rozdíly v sekvencích nukleotidů mezi jedinci stejného původu jako i napříč různými druhy. Analýza DNA hraje podstatnou roli mimo jiné při třídění organizmů, ve forenzních vědách a v neposlední řadě i v medicíně. Pomáhá např. v diagnóze nemocí, výzkumu nových léků, nebo k uzpůsobení léčby a terapie konkrétního člověka podle jeho genetických atributů.

Pro všechny tyto účely je vhodné, až nezbytné, nashromáždit velké množství biologických dat. Jejich sběr často probíhá na molekulární úrovni a dosahuje značného objemu. Při uchovávání velkého množství dat se časem narážíme na fyzické omezení paměťového prostoru. Zmenšení objemu dat, bez ztráty obsažených informací, umožňuje účinnější využití úložiště a potažmo snížení datového přenosu.

Záměr této práce je ponořit se do problematiky komprese DNA. Zmapovat existující nástroje a používané postupy. Dále udělat vlastní návrh, provést jeho implementaci a porovnat výsledky komprese s ostatními programy.

K pochopení specifik genetických sekvencí je nutné mít základní náhled do molekulární biologie. Kapitola 1 obecně popisuje DNA, její stavbu a vlastnosti, které jsou relevantní k záměru této práce. Dále se věnuje propojení biologie s informatikou. Kapitola 2 se zabývá kompresí obecně. Vysvětluje základní pojmy a rozebírá vybrané univerzální metody, jejichž principů často využívají i pokročilejší metody. V kapitole 3 popisuji existující algoritmy a nástroje specializované pro kompresi DNA sekvencí. Návrh a popis implementace vlastní metody je uveden v kapitole 4. V kapitole 5 následuje porovnání výsledků komprese nad používanou sadou souborů. V přehledové tabulce jsou uvedeny hodnoty kompresních poměrů od programů určených čistě pro DNA sekvence, jakož i některých univerzálních kompresních nástrojů. Závěr obsahuje shrnutí této práce.

1 DNA

Jak již bylo zmíněno v úvodu, role nukleové kyseliny DNA je pro život naprosto klíčová. Jedná se o hlavní zdroj genetické informace pro každou buňku. Obsahuje kód, podle kterého se buňka vyvíjí. Dalo by se očekávat, že její obsah není neuspořádaný nebo náhodný a bude obsahovat pravidelnosti.

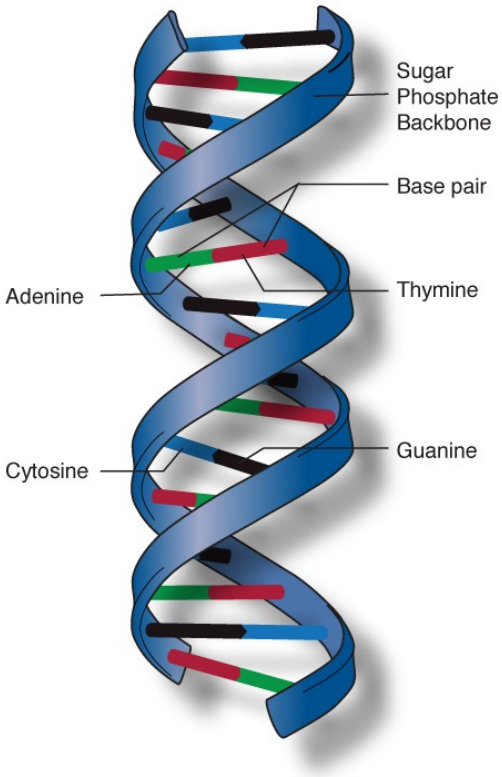
Kompletní DNA sekvenci živého organizmu nazýváme *genom*. Základní jednotkou dědičné informace jsou *geny*. Jde o úseky DNA, které jsou zdrojem informací pro syntézu proteinů. Proteiny jsou podstatou všech živých organismů. Gen předepisuje jak protein vyrobit. Obvykle obsahuje jeden gen návod na sestavení jednoho proteinu. Stejný gen je v DNA uložen redundantně. Duplikuje se na více míst biologické sekvence z evolučních důvodů, jako záloha. Je známo, že se v DNA sekvencích (zvláště u hierarchicky vyšších eukaryotních/jaderních organizmů) často vyskytují některé části řetězce opakovaně. Okolo 45% lidského genomu se skládá z opakovaných sekvencí různé délky [1].

Výše uvedené informace by mohly naznačovat, že obsah DNA bude vhodný ke kompresi – přepsán do zpětně čitelné podoby bez opakujících se informací (společných částí stavebních schémat nebo celých záloh genů). Nicméně tyto pravidelnosti jsou narušovány *mutacemi*. Mutace jsou změny v genotypu organismu oproti původní předloze. Pokud k nim dochází během procesu replikace DNA, bez zásahu z vnějšího prostředí, nazýváme je spontánní. I díky samoopravným mechanizmům se pravděpodobnost takové chyby pohybuje v řádech okolo 10^{-7} [2]. Většina mutací je indukovaných, tedy vyvolaných vnějšími faktory (mutageny fyzikální, chemické nebo biologické). Mutacemi vznikají nové změny. Ty mohou být pro organismus nevýhodné a zapříčinit různá onemocnění, nebo mohou naopak být startovními body evoluce.

1.1 Struktura

Primární strukturu DNA tvoří dlouhý lineární řetězec. DNA je směrovaná, lze jednoznačně odlišit oba konce a podle zavedené konvence určit její začátek. Jejími základními stavebními prvky jsou nukleotidy. Skládají se z cukru (deoxyribóza), fosfátu (zbytek kyseliny fosforečné) a jedné ze čtyř dusíkatých bází. Tyto báze jsou *Adenin* (A), *Guanin* (G), *Cytosin* (C) a *Tymin* (T). Podle společného molekulového základu je můžeme dělit na deriváty purinu (A, G) a pyrimidinu (C, T).

Stavbu DNA lze studovat z více úrovní. Nad první úrovní lineárního řetězce, existuje sekundární struktura typicky v podobě pravotočivé dvoušroubovice (double helix). Není to jediná formace této úrovně, ale vyskytuje se drtivé většině případů.



Obrázek 1.1 Dvoušroubovice DNA -cukr s fosfátem tvoří páteř vlákna, protilehlé báze se spojují do párů [Převzato z National Human Genome Research Institute [4]]

Dvoušroubovice je tvořena dvěma lineárními řetězci neboli vlákny. Stabilitu zajišťují patrové interakce a příčné vazby - vodíkové můstky. Ty vznikají mezi bázemi, kdy se páruje A – T a G – C. V obou případech jde o spojení jedné báze purinu a jedné báze pyrimidinu. Tento jev se nazývá *komplementarita bází* a z ní vychází i vzájemná komplementarita obou vláken DNA. Vlákna jsou vůči sobě v opačném směru (antiparalelní) a navzájem si jsou negativem. Polovina "žebříku" tak tvoří šablonu (templát) pro znovuvytvoření druhé poloviny DNA při její replikaci. O dvou protilehlých spojených bázích mluvíme jako o komplementárním páru bází, označujeme jako *base pair* (zkratka *bp*). V DNA je stejný počet bází A a T, to samé platí i pro počet bází G a C. Tyto vztahy jsou známy jako *Chargaffova pravidla*. Poměr počtu A+T a G+C se liší napříč organismy a není pro něj žádná konstanta [1].

$$\frac{|A|+|C|}{|T|+|G|} = 1 \qquad (\mathbf{1}.1)$$

Rovnice vyjadřuje poměr počtů bází ve dvoušroubovici DNA [2].

Jednotlivé nukleotidy se navzájem od sebe liší pouze přivěšenou bází. To znamená, že veškerou genetickou informaci konkrétního organizmu lze vyjádřit jako posloupnou sekvenci symbolů ze čtyř prvkové abecedy {A, C, G, T}. DNA může být uložena v podobě jednoduchého textového souboru.

```
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGC
CCTGCCCCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATA
AGGAAAAGCAGCCTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCG
GGCCCCTCATAGGAGAGGAAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCA
ATCCGCGCGCCGGGACAGAATGCCCTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAA
TAAAACCTCACCCATGAATGCTCACGCAAGTTTAATTACAGACCTGAA
```

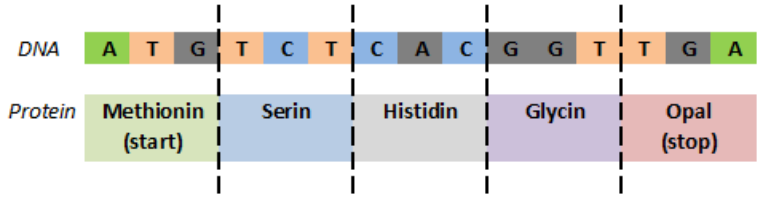
Obrázek 1.2 Ukázka DNA sekvence člověka (Homo Sapiens, plain sequence format)[7]

1.2 Kódující DNA

Sekvenci DNA, která obsahuje geny – je přepisována na proteiny, nazýváme kódující. Jednotlivé báze představují "písmena" genetického kódu. Slova jsou tvořena sekvencí o délce 3 znaků (kód je tripletový), nazýváme je *kodon*. Slov, kterých je možné zapsat pomocí třech písmen z čtyř prvkové množiny je 64 (4^3). Podle kodonů lze vytvořit všechny známé aminokyseliny. Některé mají pouze jeden kód (např. *Tryptofan* "TGG"), jiné jich mají až 6 (např. *Leucin* "TTA", "TTG", "CTT", "CTC", "CTA", "CTG"). Protein je tvořen sekvencí těchto aminokyselin, speciální kodony určují jejich začátek a konec.

první báze	*druhá báze*								*třetí báze*
	T		C		A		G		
T	TTT	Fenylalanin	TCT	Serin	TAT	Tyrosin	TGT	Cystein	T
	TTC	Fenylalanin	TCC	Serin	TAC	Tyrosin	TGC	Cystein	C
	TTA	Leucin	TCA	Serin	TAA	Ochre (stop)	TGA	Opal (stop)	A
	TTG	Leucin	TCG	Serin	TAG	Amber (stop)	TGG	Tryptofan	G
C	CTT	Leucin	CCT	Prolin	CAT	Histidin	CGT	Arginin	T
	CTC	Leucin	CCC	Prolin	CAC	Histidin	CGC	Arginin	C
	CTA	Leucin	CCA	Prolin	CAA	Glutamin	CGA	Arginin	A
	CTG	Leucin	CCG	Prolin	CAG	Glutamin	CGG	Arginin	G
A	ATT	Isoleucin	ACT	Threonin	AAT	Asparagin	AGT	Serin	T
	ATC	Isoleucin	ACC	Threonin	AAC	Asparagin	AGC	Serin	C
	ATA	Isoleucin	ACA	Threonin	AAA	Lysin	AGA	Arginin	A
	ATG	Methionin (start)	ACG	Threonin	AAG	Lysin	AGG	Arginin	G
G	GTT	Valin	GCT	Alanin	GAT	Kys. asparagová	GGT	Glycin	T
	GTC	Valin	GCC	Alanin	GAC	Kys. asparagová	GGC	Glycin	C
	GTA	Valin	GCA	Alanin	GAA	Kys. glutamová	GGA	Glycin	A
	GTG	Valin	GCG	Alanin	GAG	Kys. glutamová	GGG	Glycin	G

Obrázek 1.3 Tabulka kódu nukleotidů pro tvorbu aminokyselin



Obrázek 1.4 Ukázka přepisu DNA na protein

1.3 Nekódující DNA

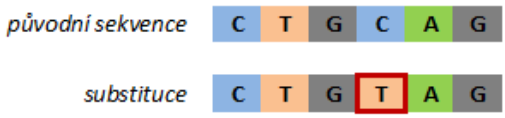
Stejně jako kódující DNA může být i nekódující DNA unikátní anebo se nacházet v genomu ve více identických či podobných kopiích. Obsahuje předpisy mechanizmů pomáhajících při replikaci DNA, také mnoho regulačních funkcí a dále úseky, o kterých genetici mluví jako o odpadní DNA ("junk DNA"). Její význam nám ještě není zcela známý, ale její odstranění má obvykle závažné negativní následky, proto nesmí být ani tato část biologické sekvence ignorována.

Podíl kódující a nekódující DNA se napříč organizmy značně liší. U bakterií se nekódující část pohybuje zhruba okolo 2% z celkové délky genomu, u eukaryot to bývá podstatně větší část. U člověka dosahuje nekódující část až 97% [2]. Statisticky tedy dochází u člověka nejčastěji k mutacím v těch místech, které nekódují proteiny. Tyto změny jsou z pohledu vývoje druhu neutrální, nemají negativní ani pozitivní účinek. Přítomnost sekvencí nekódující DNA tak snižuje dopad mutací.

1.4 Základní typy mutací

Podle způsobu vzniku můžeme popsat následující druhy mutací v DNA sekvencích:

- Substituce - Náhrada původní báze za jinou. V případě záměny v rámci původu derivátu, tedy purin-purin (A, G) nebo pyrimidin-pyrimidin (C, T), mluvíme o *tranzici*. V případě křížové náhrady jde o *transverzi*. Ačkoli je možných kombinací transverzí dvakrát více, molekulární mechanizmy způsobují větší pravděpodobnost četnosti tranzicím.

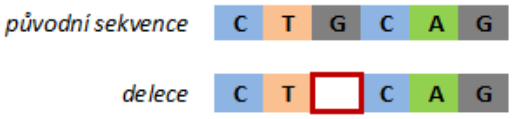


Obrázek 1.5 Příklad substituce (tranzice)

- Inzerce - Inzerce (nebo také adice), je vložení jednoho nebo více nukleotidových párů. Pokud je jejich počet jiný než násobek tří, dochází k zásadní změně při čtení kodonů. Posouvá se čtecí rámec, čímž se zvyšuje dopad mutace. Tento stav často vede ke vzniku zkráceného nebo nefunkčního proteinu.

Obrázek 1.6 Příklad inzerce

- Delece - Druh mutace, kdy dochází ke ztrátě části sekvence. Stejně jako u inzerce může vést k posunu čtecího rámce.



Obrázek 1.7 Příklad delece

1.5 Bioinformatika

Lidský genom obsahuje přes 3 miliardy párů bází. Sekvence DNA dvou jedinců stejného druhu si jsou navzájem velmi podobné, statisticky se liší pouze asi jedním prvkem z tisíce. Dva náhodně vybraní lidé mají pravděpodobnost 99,9% shody genomu [1]. Pro efektivní zkoumání takových odlišností ve velkém objemu dat je nutné nasadit techniky a metody výpočetní techniky. Strojové čtení a zpracování nám umožňuje nalézt i rozdíly, které nemusí být na první pohled vidět.

Bioinformatika zahrnuje techniky z aplikované matematiky, informatiky, statistiky a biochemie k řešení biologických úloh. Řetězec DNA je jedním z typických zástupců dat, se kterými tato vědní disciplína pracuje. Zaměřuje se např. na nacházení genů, zarovnávání sekvencí a analýzu jejich podobností.

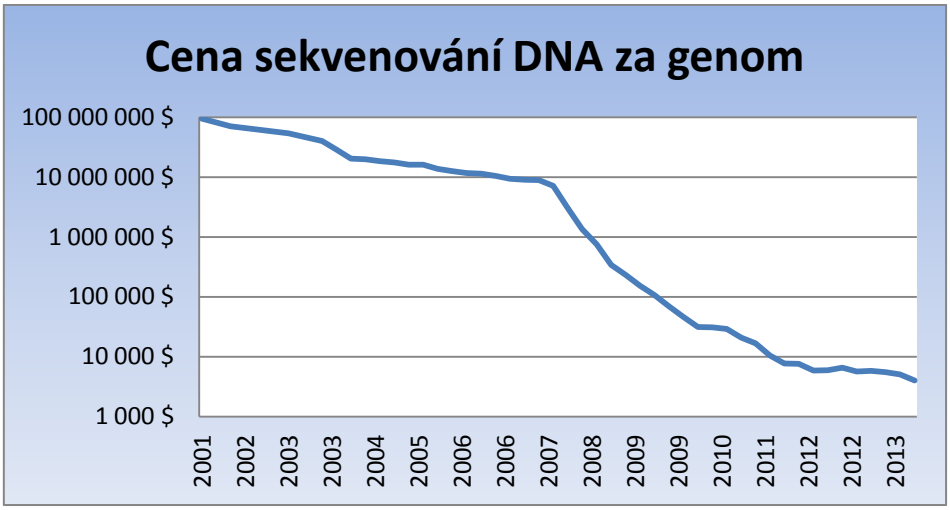
1.5.1 Genetické banky

Ve světě existují výzkumné ústavy, jako např.: The National Center for Biotechnology (GenBank)[1], European Molecular Biology Laboratory (EMBL)[2], DNA Data Bank of Japan (DDBJ)[3]. Tyto organizace mj. shromažďují a sdílejí genomické informace, fungují jako veřejně přístupné databáze. Česká republika se stala 3. června 2014 řádným členem EMBL a získala tak přístup k nejnovějším technologiím pro výzkum v oblasti genetiky, biologie a dalších příbuzných oborů [6].

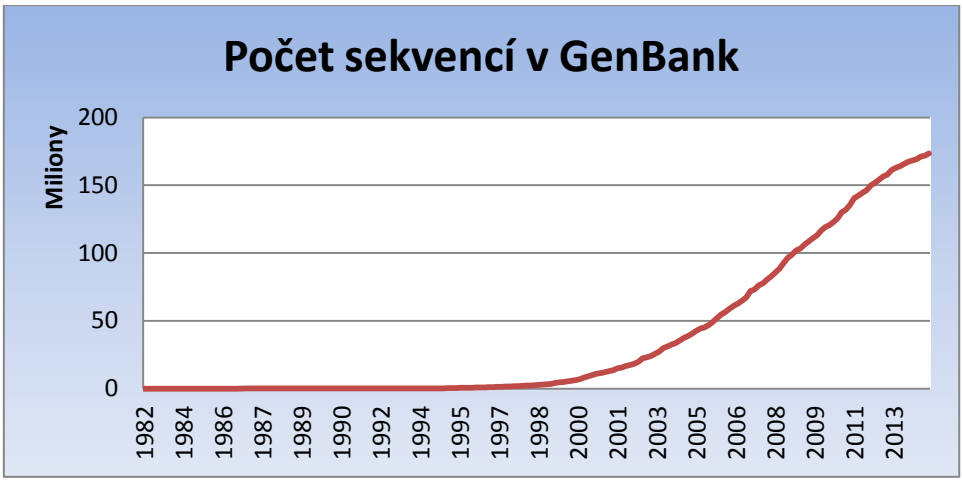
1.5.2 Sekvenování DNA

Zjišťování pořadí nukleových bází v sekvencích DNA nazýváme *sekvenování DNA*. Tento proces produkuje ohromné množství biologických dat. Velikost genomu se mezi druhy značně odlišuje. U virů je velikost běžně v řádu tisíců bází, u bakterií je v miliónech. Kompletní lidská DNA v řádu 10^9 není nejdelší. Některé druhy obojživelníků a kvetoucích rostlin se pohybují v řádech 10^{11} párů bází. S vyspělejšími a dostupnějšími technologiemi jako je *Next-generation sequencing* (*NGS*) produkce biologických dat dále nezadržitelně roste. Cena za sekvenování jednoho lidského genomu klesla pod hranici 10000$ v roce 2011.

[1] http://www.ncbi.nlm.nih.gov/
[2] http://www.embl.de/
[3] http://www.ddbj.nig.ac.jp/



Obrázek 1.8 Vývoj nákladů na sekvenování jednoho genomu (2001-2014) [zdroj dat: http://www.genome.gov/pages/der/sequencing_costs.xlsx]



Obrázek 1.9 Graf růstu uloženého počtu sekvencí v historii GenBank (1982-2014) [zdroj dat: http://www.ncbi.nlm.nih.gov/genbank/statistics]

1.5.3 Formáty souborů DNA sekvencí

Kromě jednoduché/prosté podoby uložení sekvence DNA (*plain sequence format*, Obrázek 1.2, stránka 13), jsou rozšířené i další formáty [7]. Všechny umožňují uložení více sekvencí do jednoho souboru.

- FASTA formát – začíná znakem ">" a jednořádkový popisem, který obsahuje název sekvence, identifikátor sekvence (acc), popis (descr) a délka v bp (len). Poté následuje obsah sekvence.

```
>AB000263 |acc=AB000263|descr=Homo sapiens|len=368
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC
CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG
AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC
CTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG
TTTAATTACAGACCTGAA
```

Obrázek 1.10 Ukázka formátu FASTA

- GenBank formát – je uvozen slovem "LOCUS" a popisem dat na více řádcích. Začátek sekvence oznamuje "ORIGIN", následují řádky obsahující číslo pozice prvního znaku a samotný obsah. Ten je dělen do 6 bloků po 10 znacích. Konec oznamuje dvojice lomítek ("//").

```
LOCUS       AB000263                368 bp    mRNA    linear   PRI 05-FEB-1999
DEFINITION  Homo sapiens mRNA for prepro cortistatin like peptide, complete
            cds.
ACCESSION   AB000263
ORIGIN
        1 acaagatgcc attgtccccc ggcctcctgc tgctgctgct ctccggggcc acggccaccg
       61 ctgccctgcc cctggagggt ggccccaccg gccgagacag cgagcatatg caggaagcgg
      121 caggaataag gaaaagcagc ctcctgactt tcctcgcttg gtggtttgag tggacctccc
      181 aggccagtgc cgggcccctc ataggagagg aagctcggga ggtggccagg cggcaggaag
      241 gcgcaccccc ccagcaatcc gcgcgccggg acagaatgcc ctgcaggaac ttcttctgga
      301 agaccttctc ctcctgcaaa taaaacctca cccatgaatg ctcacgcaag tttaattaca
      361 gacctgaa
//
```

Obrázek 1.11 Ukázka formátu GenBank

- EMBL formát – obsahuje na prvním řádku "ID" a identifikátor řetězce. Další řádky také obsahují dvoumístný kód a hodnotu atributu. Obsah začíná po řádku s kódem "SQ" a je strukturován podobně jako v případě formátu GenBank. Záznam je ukončen dvojicí lomítek ("//").

```
ID   AB000263 standard; RNA; PRI; 368 BP.
XX
AC   AB000263;
XX
DE   Homo sapiens mRNA for prepro cortistatin like peptide, complete cds.
XX
SQ   Sequence 368 BP;
     acaagatgcc attgtccccc ggcctcctgc tgctgctgct ctccggggcc acggccaccg        60
     ctgccctgcc cctggagggt ggccccaccg gccgagacag cgagcatatg caggaagcgg       120
     caggaataag gaaaagcagc ctcctgactt tcctcgcttg gtggtttgag tggacctccc       180
     aggccagtgc cgggcccctc ataggagagg aagctcggga ggtggccagg cggcaggaag       240
     gcgcaccccc ccagcaatcc gcgcgccggg acagaatgcc ctgcaggaac ttcttctgga       300
     agaccttctc ctcctgcaaa taaaacctca cccatgaatg ctcacgcaag tttaattaca       360
     gacctgaa                                                               368
//
```

Obrázek 1.12 Ukázka formátu EMBL

2 Komprese

2.1 Základní pojmy

Komprese dat je proces proměny vstupního (původního) datového proudu do výstupního (nového) datového proudu o menší velikosti [8]. Datovým proudem je soubor nebo vyrovnávací paměť.

Způsob reprezentace dat v souboru nazýváme *kódováním*. Při kompresi transformujeme způsob kódování informací do efektivnější podoby. Měřítkem úspěšnosti komprese může být *kompresní poměr* zapsán následujícím vzorcem (2.1).

$$Kompresní\ poměr = \frac{velikost\ výstupních\ dat}{velikost\ vstupních\ dat} \qquad (\mathbf{2}.1)$$

Pokud má původní soubor velikost např. 10 a po komprimaci dosahuje velikosti 3, vychází hodnota kompresního poměru na 0,3. Tato hodnota znamená, že se stejná data, ale v jiné reprezentaci, podařilo uložit pouze za využití 30% místa původní velikosti. Čím nižší je hodnota kompresního poměru, tím lepší efektivity reprezentace dat je dosáhnuto. Hodnoty větší než 1 znamenají negativní kompresi - výstupní soubor je větší než vstupní.

Kompresní poměr vyjádřený v rovnici (2.1) je hodnotou v procentech nebo v jednotkách bitů na bit. V publikacích se často vyskytuje vyhodnocení kompresního poměru v počtu *bitů na bajt* (*bits per byte*, zkratka *bpb*). Jeden bajt obsahuje 8 bitů, proto výše uvedený příklad kompresního poměru 0,3 je roven 2,4 bpb. V oblasti komprese biologických sekvencí jsou uváděny hodnoty v počtu bitů na bázi (*bits per base*, taktéž zkratka *bpb*).

Dalším používaným vyjádřením efektivity kódování je i veličina zvaná *kompresní faktor*. Jedná se o inverzní funkci ke kompresnímu poměru. Větší faktor znamená větší kompresi.

Mějme data, která se skládají z n různých symbolů $a_1, a_2, \dots, a_n$ s jejich pravděpodobností výskytu $P_1, P_2, \dots, P_n$. Pak e*ntropie* H_i udává množství informace, která je reprezentována symbolem a_i.

$$H_i = -\log_2 P_i \qquad (\mathbf{2}.2)$$

Entropie H_i je tím větší, čím menší je pravděpodobnost P_i výskytu symbolu a_i. Jinými slovy se dá říct, že informace o výskytu symbolu je významnější, když je jeho pravděpodobnost objevení ve zprávě menší.

Střední entropii H udává množství informace ve zprávě. Někdy je popisována jako míra neuspořádání zprávy nebo míra neurčitosti. Zprávu nelze beze ztráty komprimovat více, než právě uvádí hodnota entropie. Lze ji vypočíst jako sumu entropií jednotlivých symbolů vynásobenou pravděpodobností tohoto symbolu (vzorec 2.3).

$$H = P_1 H_1 + P_2 H_2 + \cdots + P_n H_n \qquad (\mathbf{2}.3)$$

Po dosazení H_i za jednotlivé entropie můžeme vzorec pro střední entropii upravit na následující výraz.

$$H = -\sum_1^n P_i \log_2 P_i \qquad (\mathbf{2}.4)$$

Redundance vyjadřuje větší množství informací než je nezbytné při zachování zprávy. Vychází z faktu, že střední entropie je největší, když dosahují pravděpodobnosti všech symbolů stejných hodnot. Redundance R je definována jako rozdíl mezi maximální možnou entropií zprávy a její aktuální entropií.

$$R = [-\sum_1^n P \log_2 P] - [-\sum_1^n P_i \log_2 P_i] = \log_2 n + \sum_1^n P_i \log_2 P_i \qquad (\mathbf{2}.5)$$

2.2 Kategorie

Existuje mnoho metod pro kompresi dat. Jsou založeny na různých myšlenkách a bývají vhodné pro rozdílné typy dat. Některé mají univerzální použití, jiné jsou určené pro konkrétní účely. Všechny jsou ale založeny na podobném principu – komprimují obsah odebíráním redundance z původních dat.

Z pohledu zachování původní informace dělíme metody do dvou skupin:

- *Ztrátová komprese* je také označována jako nepřesná nebo nevratná. Méně významné hodnoty jsou ztraceny k dosažení lepšího kompresního poměru. Používá se převážně při kompresi audio a video záznamů, kdy kvůli nedokonalosti lidských smyslů můžeme vypustit detaily spadající pod naši rozlišovací úroveň. Podobná situace také nastává, když reprodukční zařízení nebo nosné médium audio/vizuální informace není schopné nést vysokou preciznost (jemnost) původních dat.
- *Bezeztrátová komprese* je naopak přesná. Původní data jsme schopni při zpětném procesu dekomprese zcela rekonstruovat a dosáhnout původního stavu.

V případě komprese biologické sekvence je žádoucí zachovat původní data beze ztráty jediné informace nebo změny obsahu. V této práci se budu zabývat pouze technikami ze skupiny bezeztrátové komprese.

Podle použité strategie komprese mohou být metody z těchto kategorií:

- *Statistické metody* komprese jsou založeny na pravděpodobnosti výskytů jednotlivých symbolů. Používají variabilní délku kódu. Znakům (nebo skupině znaků) s vyšší četností je přiřazen kratší kód než těm méně pravděpodobným. Při používání různé délky kódu je důležité zvolit vhodný kód, aby bylo dekódování jednoznačné. Algoritmy obsahují část, kdy je zjišťována pravděpodobnost znaků – statistický model. Na základě modelu je znakům přiřazen kód, podle kterého je následně přepsán původní obsah.
- *Slovníkové metody* využívají opakující se části textu. Jeden z výskytů takové části je zaznamenán do speciální dynamické datové struktury – slovníku, všechny ostatní výskyty jsou nahrazeny odkazem na jeho umístění. Jedny z

nejpoužívanějších algoritmů této kategorie pocházejí od autorů Abraham Lempel a Jacob Ziv z konce 70. let minulého století.

2.3 Fibonacciho kódování

Nabízí efektivní způsob binárního kódování celých kladných čísel. Tato metoda je inspirována *Fibonacciho posloupností* (označována též jako *Fibonacciho čísla*), kde každé číslo je součtem dvou předchozích.

Fibonacciho posloupnost: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

$$F(n) = F(n-1) + F(n-2) \tag{2.6}$$

Rekurzivní definici Fibonacciho posloupnosti lze zapsat jako:

$$F(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F(n-1) + F(n-2) & n \geq 1 \end{cases} \tag{2.7}$$

Fibonacciho kódování je založeno na faktu, že každé celé nezáporné číslo může být unikátně vyjádřeno sumou Fibonacciho čísel, kdy žádné 2 po sobě jdoucí Fibonacciho čísla nejsou použita v reprezentaci.

$$V\big(F(n)\big) = n = \sum_{i=0}^{p} \mathrm{a_i F_i} \ (\mathrm{a_i} \in \{0,1\}, 0 \leq \mathrm{i} \leq \mathrm{p}) \tag{2.8}$$

Rovnice vyjadřuje Fibonacciho binární kódování a zároveň výpočet jeho hodnoty, kde $F(n) = a_0 a_1 a_2 \ldots a_p$ je Fibonacciho binární kódování celého kladného čísla n.

Tímto způsobem vyjádříme kódované číslo v binárním zápisu, kdy bity budou mít postupně váhu čísla z Fibonacciho posloupnosti (1, 2, 3, 5, 8, …). Nikdy nejsou dva následující bity nastaveny na jedna současně. Přidáním dalšího bitu s hodnotou 1 za bit s nejvyšší váhou jednoznačně definujeme konec kódu pro dané číslo.

Tabulka 2.1 *Fibonacciho kódování celých kladných čísel*

hodnota	*váha bitu*						*Kód*	*Fibonacciho kód*
	1	*2*	*3*	*5*	*8*	*…*		
1	1	0	0	0	0	0	1	11
2	0	1	0	0	0	0	01	011
3	0	0	1	0	0	0	001	0011
4	1	0	1	0	0	0	101	1011
5	0	0	0	1	0	0	0001	00011
6	1	0	0	1	0	0	1001	10011
7	0	1	0	1	0	0	0101	01011
8	0	0	0	0	1	0	00001	000011

V případě, kdy častěji kódujeme menší čísla, lze Fibonacciho kódování transformovat do vhodnější podoby. Mluvíme o *k-posunutém Fibonacciho kódování* (*k-Shifted Fibonacci*

encoding). Tato metoda při kódování hodnot $n \in \{1, \dots, 2^k - 1\}$ používá klasický binární kód. Ten musí mít dostatečnou délku, aby pokryl $2^k - 1$ hodnot (např.: pro $k = 1 \rightarrow \{1\}$, pro $k = 2 \rightarrow \{01, 10, 11\}$, pro $k = 3 \rightarrow \{001, 010, 011, 100, \dots\}$). Pro hodnoty $n \geq 2^k$ se používá konstantní prefix 0 od délce k (např.: pro $k = 1 \rightarrow 0$, pro $k = 2 \rightarrow 00$), za kterým následuje Fibonacciho kód pro $n - (2^k - 1)$.

Tabulka 2.2 *k-posunuté Fibonacciho kódování, ukázka kódu*

kódování	*hodnota pro n*							
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Fibonacci	11	011	0011	1011	00011	10011	01011	00011
1-shifted Fib. ($k = 1$)	1	011	0011	00011	01011	000011	010011	001011
2-shifted Fib. ($k = 2$)	01	10	11	0011	00011	000001	001011	0000011
3-shifted Fib. ($k = 3$)	001	010	011	100	101	110	111	00011

Modrá barva znamená použití binárního kódování, červená barva vyznačuje konstantní prefix, černá je pro původní Fibonacciho kód.

Tabulka 2.3 *k-posunuté Fibonacciho kódování, délka kódu*

kódování	*délka kódu pro n*							
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Fibonacci	2	3	4	4	5	5	5	5
1-shifted Fib. ($k = 1$)	1	3	4	5	5	6	6	6
2-shifted Fib. ($k = 2$)	2	2	2	4	5	6	6	7
3-shifted Fib. ($k = 3$)	3	3	3	3	3	3	3	5

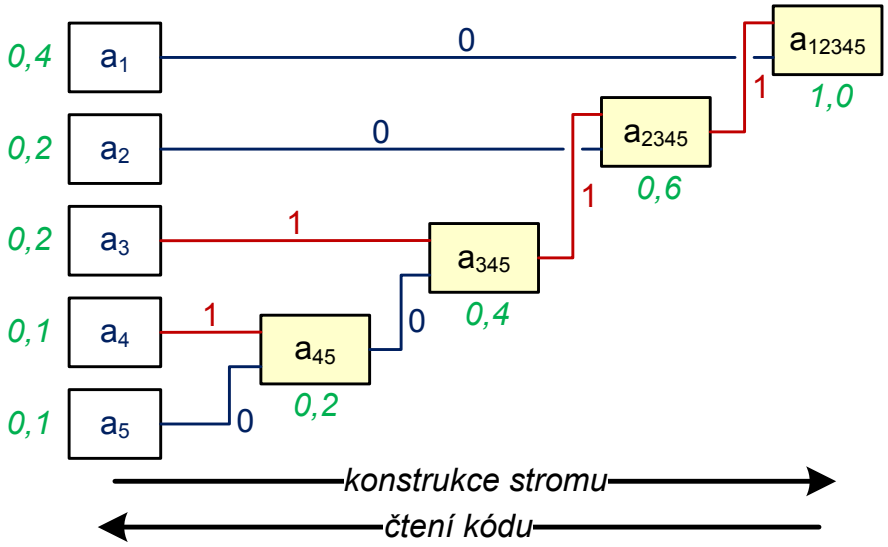
Všechny tyto metody jsou prefixové. To znamená, že žádný kód není prefixem (předponou) jiného kódu. Výhodou prefixového kódování je jednoznačné dekódování. Při čtení kódu zleva doprava odebíráme bity, dokud nedostaneme kód některého ze symbolů. Mezi symboly není nutné zapisovat speciální oddělovače a tímto dosahují metody tohoto typu úspory v celkovém počtu použitých bitů.

2.4 Huffmanovo kódování

Huffmanovo kódování patří k nejstarším metodám, pochází už z roku 1952. Jde asi o nejznámější statistické kódování. Vychází z tabulky četností symbolů a vytváří prefixové kódy proměnné délky. Má hodně společného s *Shannon-Fano kódováním*. Rozdíl je ve směru konstrukce kódovacího stromu, které *Huffmanovo kódování* buduje odspodu. Komprese i dekomprese je velmi rychlá a nemá příliš velké nároky na paměť. Nejlepších výsledků dosahuje při pravděpodobnostech symbolů rovných mocninám 2, poté se kódování blíží entropii.

Nejdříve jsou zjištěny četnosti všech symbolů vstupní abecedy. Následuje výpočet jejich pravděpodobnosti, která je dána poměrem četnosti symbolu a celkové délky kódovaných dat. Symboly seřadíme podle vypočtené pravděpodobnosti sestupně. Začínáme odspodu budovat binární strom, kde jsou symboly zapsány na listech stromu. Vybereme dva symboly

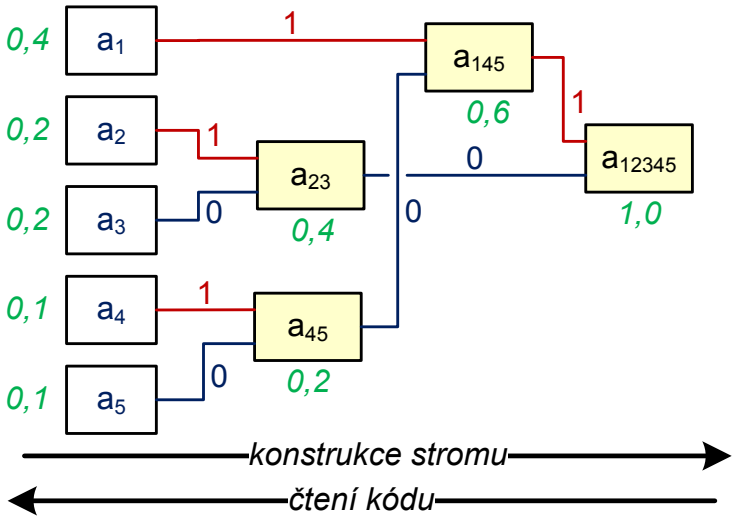
s nejmenší pravděpodobností a sloučíme je do nového symbolu (jeho pravděpodobnost je rovna součtu původních symbolů) tak, aby představoval nový uzel na vyšší úrovni stromu. Tento krok opakujeme, dokud nedosáhneme pouze jednoho symbolu/kořene s pravděpodobností 1. Hranám stromu přiřazujeme binární hodnoty {0,1}. Dalším krokem je čtení vytvořených kódů pro všechny symboly vždy od kořene až k listu (které tyto symboly označují). Konstrukci binárního stromu a následně kódu znázorňuje příklad (Obrázek 2.1). Pomocí takto vytvořené tabulky kódů pak probíhá samotná komprese. Ze vstupu jsou čteny symboly, které jsou poté příslušným kódem zapsány do výstupního souboru na binární úrovni.



symbol	*pravděpodobnost*	*kód*
a_1	0,4	0
a_2	0,2	10
a_3	0,2	111
a_4	0,1	1101
a_5	0,1	1100

Obrázek 2.1 Ukázka Huffmanova kódování

Pokud dosahuje více symbolů (původní i vytvořené spojením) stejných pravděpodobnostních hodnot, existuje více způsobů jak binární strom vybudovat. Stejně tak je i více možností jak přiřadit hranám binární hodnoty. Následuje příklad (Obrázek 2.2), kde je vytvořen odlišný strom pro stejný příklad (Obrázek 2.1). V jednom kroku došlu k rozdílu ve výběru dvou jiných symbolů z možných 3 o stejné pravděpodobnosti, což vedlo k vygenerování odlišného kódu. V druhém případě dochází k menší odchylce v délkách kódů. Tato varianta je preferována pro stabilnější frekvenci plnění mezipaměti (*buffer*) při přenosu komprimovaného proudu.



symbol	pravděpodobnost	kód
a_1	0,4	11
a_2	0,2	01
a_3	0,2	00
a_4	0,1	101
a_5	0,1	100

Obrázek 2.2 Ukázka Huffmanova kódování - druhá varianta

Pro dekompresi je nutné do zakódovaného souboru zapsat použitou tabulku kódů. Také lze použít pouze údaje o pravděpodobnostech jednotlivých symbolů a poté binární strom opět zrekonstruovat, až dojdeme k tabulce kódů. Musíme však zajistit dodržení pravidel, aby byla tvorba stromu pro kompresi i dekompresi jednoznačná.

Dekódování poté probíhá odebírání binárních znaků ze vstupu, dokud se přečtený kód nerovná některému z kódovací tabulky. Pak odpovídající symbol zapíšeme na výstup. Takto postupujeme, dokud nepřeložíme celý vstupní soubor.

Místo pravděpodobnosti symbolů můžeme použít jejich počet výskytů (frekvenci). Počet výskytů je přímo úměrný pravděpodobnosti, takže potřebné informace pro kódování jsou zachovány a je výhodnější pracovat s celými čísly.

2.5 Aritmetické kódování

Princip Aritmetického kódování poprvé navrhnul Peter Elias na začátku šedesátých let 19. století. Tento způsob statistické metody je početně náročnější a pomalejší než Huffmanovo kódování, ale dosahuje lepších výsledků. Umí lépe zakódovat nerovnoměrné rozložení četností.

Základní myšlenkou je zakódování celé zprávy do jednoho desetinného čísla z intervalu $\langle 0, 1)$. Tento interval je v každém kroku přerozdělen na takový počet podintervalů, který odpovídá velikosti vstupní abecedy. Intervaly nejsou přidělovány rovnoměrně, ale na

základě vypočtené pravděpodobnosti daného symbolu. To znamená, že intervalu, který odpovídá symbolu s častějším výskytem, bude přiřazeno širší pásmo.

$$I = \langle I + q_{i-1} * (h - l), I + q_i * (h - l) \rangle \qquad (2.9)$$

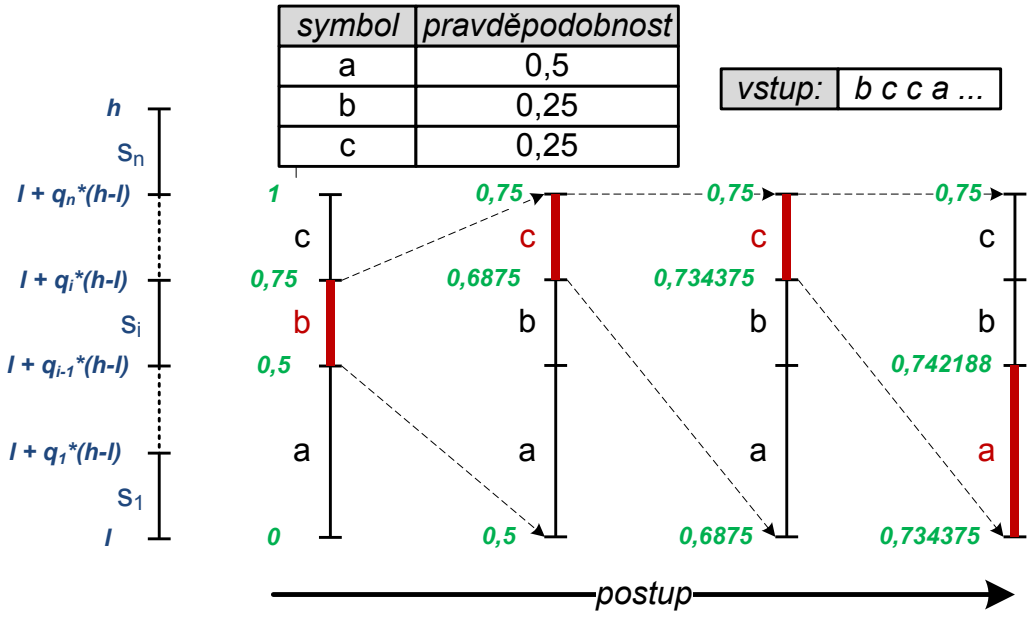
$I = \langle l, h \rangle$ stávající hodnoty intervalu (l dolní, h horní hodnota)

q_{i-1} kumulativní pravděpodobnost předchozího symbolu

q_i kumulativní pravděpodobnost aktuální symbolu

Interval I pro daný symbol je přepočítáván podle vzorce (2.9). Kumulativní pravděpodobnost symbolu je dána součtem vlastní pravděpodobnosti (P_i) a pravděpodobností všech předchozích symbolů ($P_{i-1} + P_{i-2} + \cdots + P_1$), kterým již byl interval přirazen (začínaje od nuly $\langle 0, P_1 \rangle$).

Kódování zprávy postupně vybírá desetinné číslo tak, aby při každém přečtení symbolu prošlo vnitřkem odpovídajícího intervalu v daném kroku. Výstupní číslo bude růst průchodem užšími intervaly (reprezentující méně časté symboly) rychleji, kvůli nutnosti vyšší preciznosti v menším rozpětí. Čím menší je interval, tím delší jsou čísla v jeho zápisu. Tímto způsobem je dosažena podstata komprese – symboly s vyšší pravděpodobností budou zakódovány do kratšího kódu než symboly méně časté. Do komprimovaného souboru jsou zapsány informace o pravděpodobnostech symbolů, délka vstupní zprávy a výstupní desetinné číslo. Postup procházení intervalů a jejich přepočet znázorňuje příklad (Obrázek 2.3).



Obrázek 2.3 Ukázka Aritmetického kódování

Při používání tohoto kódování narážíme na limity použitých datových typů, které jsme při implementaci nasadili. Pokud dojde k dosažení stavu tak úzkého intervalu, že již nejde dále

rozdělit, tak se desetinné číslo z tohoto intervalu se uloží na výstup a intervaly se resetují. Kódování pak pokračuje dál podle standardního postupu.

Dekomprimace probíhá obdobným způsobem. Znaky se dekódují ve stejném pořadí, v jakém byly kódovány. V každém kroku se opět přerozdělí intervaly podle uložených pravděpodobností jednotlivých symbolů. Výběrem podintervalu, který odpovídá desetinnému číslu, identifikujeme aktuální symbol a ten zapíšeme na výstup. Vše se opakuje, dokud algoritmus neinterpretuje výstupní zprávu o délce, která byla uložena v komprimovaném souboru. Bez zachování informace o vstupní délce by metoda neměla jak určit, kdy má svůj chod ukončit. Jiné možné řešení pro ukončení algoritmu je v přidání symbolu oznamující konec souboru (*EOF*) do vstupní abecedy již při procesu komprese. Tomuto speciálnímu symbolu je pak určen interval stejně jako ostatním znakům.

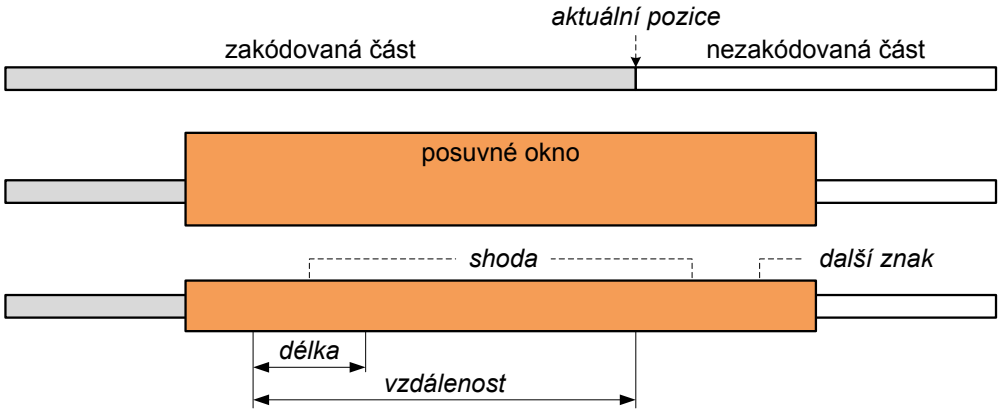
Provádění operací s desetinnými čísly (se značnou délkou za desetinnou částkou) je náročné a pomalé. Praxi je algoritmus upraven tak, aby pracoval se celými čísly nebo čísly v binární podobě.

Často je používán tento algoritmus v adaptivní variantě. Při tomto způsobu komprese je symbol kódován na základě předchozí četnosti, potom je model pravděpodobností přepočítán. Z toho principu vycházejí další pokročilé metody jako např. *Prediction by Partial Matching* (*PPM*), který využívá pro výpočet pravděpodobnosti různé délky kontextu.

2.6 LZ77

Algoritmus *LZ77* je z kategorie slovníkových metod kompresí. Autoři Abraham Lempel a Jacob Ziv navrhli několik postupů kódování na základě adresování jednoho z opakovaných výskytů podřetězce, tento konkrétní pochází z roku 1977 a někdy je označován jako *LZ1*.

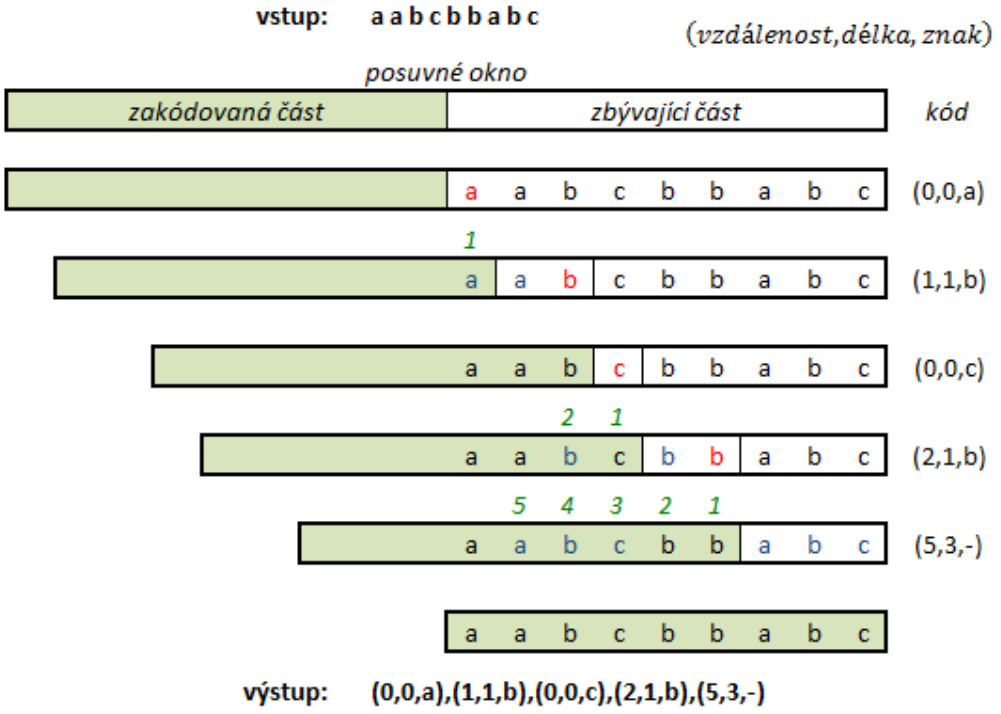
Metoda je založena na použití posuvného okna o pevné délce (*sliding window*), kterým postupně prochází celou sekvenci. Okno udržuje od aktuální pozice nalevo zakódovanou část, jeho velikost bývá v rozmezí 8-64 KB. Napravo obsahuje dosud nezakódovanou část, která je běžně o velikosti 256 B.



Obrázek 2.4 Posuvné okno v algoritmu LZ77 a vyznačení shody sekvence

Komprese

Algoritmus hledá v rámci okna nejdelší řetězec z prefixu nezakódované části s výskytem v zakódované části. Pokud nalezne více stejně dlouhých předchozích výskytů, tak vybírá ten nejbližší (který se vyskytnul poslední). Shodu nezpracované části zakóduje jako trojici: vzdálenost počátku předchozího výskytu od aktuální pozice, délku shody a první znak po shodě z nezakódované části. Pokud není shodný výskyt nalezen, jsou hodnoty vzdálenosti a délky rovny nule, první znak z nezakódované části zůstává. V dalším kroku se okno, včetně aktuální pozice, posune o délku shody a jeden znak doprava.

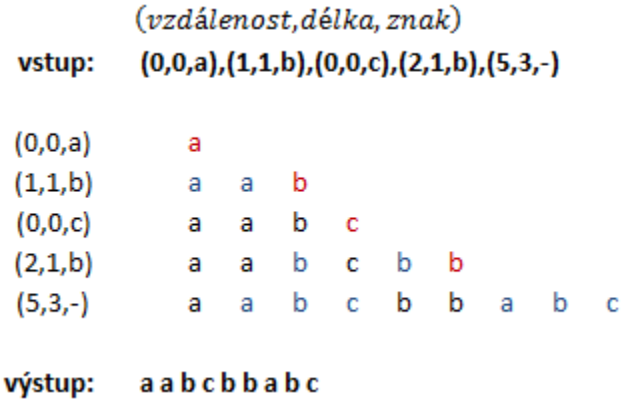


Obrázek 2.5 LZ77ukázka kódování

Na začátku procesu jsou vždy hodnoty pro vzdálenost a délku nulové, protože ještě není zakódovaný jediný znak. Každým dalším zápisem výstupní trojice bez odkazu na předchozí výskyt očividně nabývá výstupní řetězec na velikosti rychleji, než kdyby byl zapsán oznak samostatně. Také v případě, kdy je předchozí výskyt ve značné vzdálenosti od aktuální pozice, může být interpretace za pomocí trojice nevýhodná. Zápis takového opakovaného řetězce přes vzdálenost a délku nemusí vést, kvůli delšímu vyjádření vysokých čísel, k úspoře a efektivnímu kódu. Je nutné vhodně zvolit délku oknu. Dalším řešením je implementace funkce, která vyhodnotí vhodnost zakódování řetězce pomocí vzdálené nebo nulové reference.

LZ77 je asymetrická metoda. Fáze dekomprese je početně podstatně jednodušší než komprese, při které algoritmus prochází historii a hledá shodu řetězce s největší délkou. Při dekódování se pouze interpretuje přečtený kód. Pokud trojice obsahuje vzdálenost rovnající se nule, tak můžeme rovnou do výstupu zapsat znak a posunout se za jeho pozici. V jiném případě se nejdříve ve výstupním řetězci (nebo v mezipaměti obsahující výstupní data) posuneme zpět

o hodnotu vzdálenosti, podle délky postupně připisujeme nadcházející znaky od ukazatele na konec výstupu a poté zakončíme znakem z trojice s posunutím ukazatele za jeho pozici.



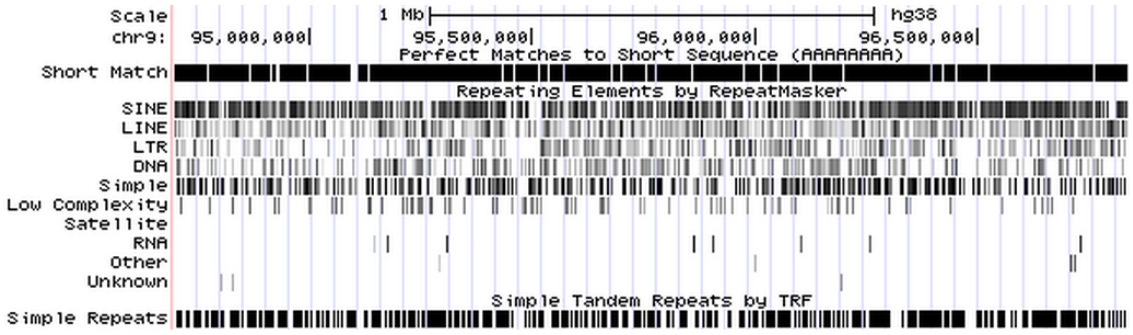
Obrázek 2.6 LZ77ukázka dekódování

3 Komprese DNA

Univerzální kompresní programy nedosahují při kódování DNA dobrých výsledků. Hlavní příčinou jejich neúspěchu je fakt, že jsou postaveny na modelech odvozených od běžného textu. Efektivní kódování DNA sekvencí je v oblasti komprese dat považováno za velmi obtížný úkol (*A new challenge for Compression Algorithms* [11]). Na první pohled jsou biologické sekvence podobné náhodným řetězcům, pravidelnosti jsou velmi skryté. Z pohledu entropie odpovídají statistickému modelu nultého stupně – každý znak vstupního textu je statisticky nezávislý na jiném znaku, pravděpodobnosti výskytu všech znaků jsou stejné. Průměrné zastoupení bází v testovacích korpusech vychází přibližně: A 30%, C 20%, G 20%, T 30% (soubor s údaji je přiložen na CD).

Komprese DNA je specifická úloha, na kterou je vhodné aplikovat zvláštní metody. Na rozdíl od univerzálních kompresních nástrojů lze využít toho, že známe dopředu vlastnosti vstupních dat. Biologické sekvence přenášejí důležité informace, které mohou být náhodně pozměněné (drobnými/bodovými) mutacemi, ale nejsou náhodné. Všechna nenáhodná data mají nějakou strukturu a pravidla, jejichž identifikace může být klíčem pro efektivnější kompresi.

Z tohoto pohledu je velmi důležitou vlastností DNA častý výskyt opakujících se řetězců. Délka těchto řetězců je velmi rozdílná, stejně jako vzdálenost jejich pozic. Části bývají delší než v běžném textu ale s menší četností. V lidském genomu je výrazný výskyt opakujících se sekvencí SINE (*short interspersed nuclear elements*) a *LINE* (*long interspersed nuclear elements*). Části SINE bývají průměrně v délce okolo 300 bází, LINE jsou běžně v délce 6000 bází. Vzhledem k velikosti abecedy DNA (pouze 4 znaky) se i v menších kontextech vyskytují často krátká opakování.



Obrázek 3.1 Analýza DNA - UCSC Genome Browser on Human Dec. 2013 (GRCh38/hg38) Assembly, zdroj: genome-euro.ucsc.edu

Grafické znázornění (Obrázek 3.1) zobrazuje analyzovaný úryvek lidské DNA o délce zhruba 2 milióny bp. Například je vidět četnost přesných výskytů krátkého řetězce `aaaaaaaa` (*Short Match*), opakující se SINE a LINE sekvence a kratší jednoduchá opakování (*Simple Repeats*).

Repetice mohou mít různou podobu:

- *přímé opakování* (*match, direct repeat*) – Případ, kdy jsou oba řetězce zcela identické.
  Např. … `ccagt` … `ccagt` …
- *Zrcadlové opakování* (*mirror/reverse repeat*) – Pořadí znaků je v jednom z řetězců obrácené.
  Např. … `ccagt` … `tgacc` …
- *Komplementární opakování* (*complementary/pairing repeat*) – Náhrada C–G a A–T v druhém řetězci. Komplementární doplněk původního řetězce (viz. Komplementarita bází v kapitole 1.1).
  Např. … `ccagt` … `ggtca` …
- *Invertované opakování (inverted/reverse-complementary repeat)* – Kombinace zrcadlení a komplementu mezi řetězci. Taktéž označováno jako *palindrom* (*palindrome*).
  Např. … `ccagt` … `actgg` …

Navíc u všech případů uvažujeme nejen o shodě přesné ale i přibližné.

V zásadě existují dva rozdílné přístupy, jak přistoupit ke kompresi DNA. První mód je označován jako „horizontální“. Pracuje pouze s jednou konkrétní sekvencí. Snaží se odhalit její vlastní redundance, v některých případech přímo celé repetice, a zkrátit její zápis. Druhý mód je nazýván „vertikální“. Jeho přístup spočívá ve využití referenčního genomu. V kapitole o DNA (1.5) jsem již zmiňoval, jak vysokou míru shody mají 2 organizmy stejného druhu. Komprimovaná sekvence je porovnávána vůči referenční, výstup poté obsahuje pouze rozdíly (ukazatele a listy editačních operace). Pro dekomprimaci je však nutné mít onen referenční genom k dispozici, nemluvě o jeho vhodné volbě pro použití s aktuální sekvencí. Tento mód komprese se rozšířil především po roce 2003, kdy byl dokončen projekt prvního zmapování celého lidského genomu *Human Genome Project* [4].

Postupy, které při kompresi sekvence používají referenční genom, mají dle mého názoru velký potenciál k dosažení vysoké efektivity. Avšak pro jejich závislost na dostupnosti jiného biologického řetězce jsem se rozhodnul pro zaměření na metody, které pracují v rámci jedné DNA sekvence.

U některých metod autoři předpokládají v praxi scénář, kdy po sekvenování DNA se soubor bude komprimovat pouze jednou. Výstupní data se nahrají do databáze, odkud budou dostupná více uživatelům. Ti si poté komprimovanou sekvenci přenesou na lokální úložiště, kde s ní budou pracovat. To znamená, že mnohem častěji bude probíhat dekomprese. Proto může být pomalá rychlost komprese tolerována, ale je všeobecně požadován nenáročný běh zpětného dekódování. Tyto metody jsou kvůli podstatně odlišné náročnosti nazývány asymetrické.

V následujících kapitolách popisuji vybrané algoritmy a programy pro kompresi DNA v chronologickém pořadí. Jejich dosažené výsledky nad standardním souborem sekvencí jsou uvedeny ve srovnávací tabulce v kapitole 5 Porovnání.

3.1 Dvoubitové kódování (Base2 Binary)

Jedná se asi o nejjednodušší způsob komprese DNA sekvence. Někdy je označováno jako „naivní“ kódování. Vychází z předpokladu, že pro zápis 4 různých znaků (bází) stačí použít pouhé 2 bity ($2^2 = 4$). V běžném textovém souboru je použito 8 bitů (jeden bajt) na jeden znak. Tzn. 256 různých hodnot (2^8), které dostatečně pokrývají rozlišení malých a velkých písmen (anglické) abecedy včetně číslic a základních znamének. Pro kódování DNA je použití celých 8 bitů na jednu bázi zbytečné. Tento způsob umožňuje do 8 bitů uložit 4 báze. Kompresní poměr dosahuje 25%, tzn. 2 bpb (bits per base).

Do algoritmu napevno zvolíme přiřazení dvoubitového kódu, které použijeme pro kompresi i dekompresi. Při výskytu neočekávaného znaku je algoritmus přerušen. Čteme postupně báze a zapisujeme odpovídající kód. Bity zapisujeme po 4 do jednoho bajtu jako znak nebo osmiciferné binární číslo.

Tabulka 3.1 *Přiřazení dvoubitového binárního kódu bázím (jedna z variant)*

báze	*hodnota*	*binární kód*
a	0	00
c	1	01
g	2	10
t	3	11

Tabulka 3.2 *Pozice a hodnota bitu v binárním kódu*

pozice bitu	8	7	6	5	4	3	2	1
hodnota bitu	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

Pro příklad použiju sekvenci:

```
c g a t g c c a t c c g a t t g
```

Dle předpisu (3) přepíšeme na bity:

```
01 10 00 11 10 01 01 00 11 01 01 10 00 11 11 10
```

Uložení do bajtu:

```
[01100011] [10010100] [11010110] [00111110]
```

Bajty představují čísla:

```
[99] [148] [214] [62]
```

V případě, kdy délka původní sekvence má po celočíselném dělení čtyřmi nenulový zbytek, bude jen část posledního bajtu kódovat vstupní data. Při tomto kódování nemáme k dispozici další (pátý) kód, kterým bychom naznačili konec sekvence. Proto je nutné pro fázi

dekomprese zachovat délku kódované sekvence nebo uložit informaci o využití posledního bajtu.

3.2 BioCompress, BioCompress-2

První program pro bezeztrátovou kompresi DNA využívající repetice v sekvenci je *BioCompress* od dvojice S. Grumbach a F. Tahi z roku 1993 [10]. Svou metodu dále vylepšili a v roce 1994 publikovali článek popisující *BioCompress-2* [11].

Autoři nejdříve aplikovali klasické metody pro kompresi textových souborů na DNA data. Jednalo se o Huffmanovo kódování, Aritmetické kódování a slovníkové Lempel-Ziv metody komprese. Všeobecně se statistické metody ukázaly být málo efektivní, nejlepších výsledků dosahovalo Aritmetické kódování druhého řádu. Vysvětlují to faktem, že pravděpodobnost výskytu znaku silně závisí na několika předchozích znacích. Kódování prvního a třetího řádu dosahovalo stejných výsledků, při čtvtém a vyšším řádu efektivita klesala.

Postavili jednoprůchodový algoritmus na základě slovníkové komprese, ovšem s větším skenovacím oknem. Detekuje přesná opakování a komplementární palindromy, které se již dříve v sekvenci vyskytly. Nahradí je číslem pozice předchozího výskytu a hodnotou, která vyjadřuje počet znaků ve shodě. Čísla je prezentována binárním nebo Fibonacciho kódováním, které avizuje pomocný bit uvnitř čísla. Za první výskyt dvou následných jedniček '11' je u binárního kódování vložena nula a ve Fibonacciho kódování jednička. Pokud se v zakódovaném čísle nevyskytují dva bity vedle sebe s hodnotou '1', pak musí jít o binární kód, neboť Fibonacci kód tuto krátkou sekvenci vždy obsahuje (na konci). Rozhodnutí o použití jednoho ze dvou kódování je prováděno dynamicky tak, aby vedlo k co nejkratšímu kódu. Neopakující se části sekvence jsou zapisovány pomocí dvoubitového kódu, v BioCompress-2 je tato metoda nahrazena výše zmiňovaným Aritmetickým kódováním druhého řádu.

3.3 Cfact

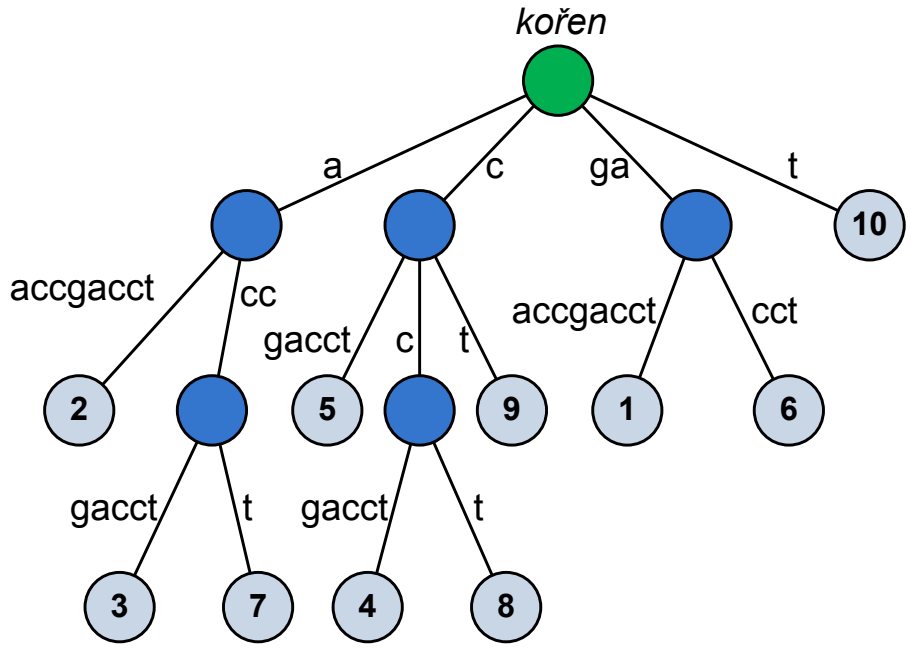
V roce 1996 vydal E. Rivals a jeho tým kompresní program *Cfact* [12]. Algoritmus má 2 fáze: rozbor dat a samotné kódování.

Program nejdříve vyhledává přesná opakování pomocí sufixového stromu. Vždy probíhá vyhodnocení, zdali by kódování nalezeného řetězce jako opakování bylo ziskové z pohledu délky kódu. Výstupem úvodního procesu je zřetězený list segmentů vybraných ke kódování.

Sufixový strom je zvláštní datová struktura, která uchová všechny přípony (sufixy) daného řetězce. Jednotlivé přípony čteme od kořene dolů. Listy (koncové uzly) obsahují číslo pozice prvního znaku přípony. Pro potřebu hledání přesných opakování je využívána ta vlastnost, že cesta od kořene dolů po vnitřních uzlech vyznačuje společné předpisy (prefixy).

Tabulka 3.3 *Příklad sekvence pro ukázku sufixového stromu*

znak	**g**	**a**	**a**	**c**	**c**	**g**	**a**	**c**	**c**	**t**
pozice	1	2	3	4	5	6	7	8	9	10



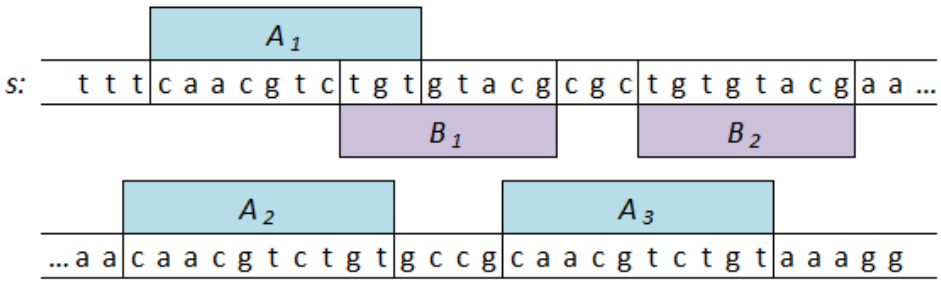
Obrázek 3.2 Ukázka sufixového stromu pro sekvenci gaaccgacct

Ve výše uvedené sekvenci je díky sufixového stromu nalezeno například opakování řetězce *acc* na pozici 3 a 7.

Výskyty nalezených opakování jsou rozděleny do tří skupin:

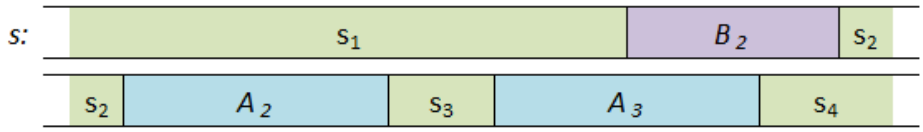
- První výskyt – označován také jako referenční. Je nejblíže začátku celé sekvence (nejvíce vlevo). Odkazují se na něj následující výskyty. Je zachován v původní podobě.
- Druhý výskyt – první nalezené opakování, je kódováno dvěmi čísly. Ukazatelem na první výskyt a délkou opakujícího se řetězce. Zároveň jsou tyto hodnoty zařazeny do slovníku.
- Další výskyty – druhé a vyšší nalezené opakování. Je nahrazeno jednou hodnotou, kterou je index do slovníku na položku druhého výskytu.

Tento speciální postup zápisu druhého výskytu vede k úspoře kódování jednoho indexu. Čísla a indexy jsou reprezentovány Fibonacciho kódováním.



Obrázek 3.3 Ukázka sekvence s vyznačenými shodami

Obrázek 3.4 zachycuje sekvenci s a v ní nalezená opakování $A = \{A_1, A_2, A_3\}$ a opakování $B = \{B_1, B_2\}$. První výskyty opakování A_1 a B_1 se mohou překrývat. Jak již bylo uvedeno, jsou v této fázi ponechány beze změny a budou se zbylými (neopakující se) částmi zahrnuty do zón.



slovník	
index_A	pos_A$_1$; |A|
index_B	pos_B$_1$; |B|

Obrázek 3.4 Ukázka rozdělení sekvence s do zón

Nyní lze sekvenci zapsat jako $s = s_1 \cdot B_2 \cdot s_2 \cdot A_2 \cdot s_3 \cdot A_3 \cdot s_4$. Současně je vytvořen slovník, jehož klíčem je index opakovaného řetězce a hodnotou je dvojice definující první výskyt (pozice, délka).

význam	*kód*		
počet zón s opakováním	3		
	typ zóny	*offset*	*ukazatel*
zóna B_2	0	$\|s_1\|$	(pos_B$_1$; |B|)
zóna A_2	0	$\|s_2\|$	(pos_A$_1$; |A|)
zóna A_3	1	$\|s_3\|$	(index_A)
zbylé zóny	dvoubitové kódování($s_1 \cdot s_2 \cdot s_3 \cdot s_4$)		

Obrázek 3.5 Tabulka obsahuje kompletní kód pro sekvenci s

Celou sekvenci s zakóduje Cfact tímto způsobem:

- První je číslo, vyjadřující počet zón kódovaných jako opakování.
- Následuje kód pro uvedený počet zón – trojice, kde první bit označuje typ zápisu ukazatele, další část označuje vzdálenost výskytu a samotný ukazatel.
- Na zbylé zóny je aplikováno dvoubitové kódování (včetně A_1 a B_1)

3.4 GenCompress

První algoritmus, který pracuje i s nepřesným opakováním [13]. Vytvořil ho X. Chen s ostatními autory v roce 1999.

Zavedli způsob zápisu editačních operací pro nepřesná opakování vzhledem k původnímu výskytu:

- *Replace / náhrada* – vyjádřeno *(R, i, znak)*. Znamená, že je původní znak na pozici *i* nahrazen *znakem* novým.
- *Insert / inzerce* – vyjádřeno *(I, i, znak)*. Na pozici *i* je vložen další *znak*.
- *Delete / delece* – vyjádřeno *(D, i)*. Znak na pozici *i* je odstraněn.

Znakem *C* vyjadřujeme v následujících příkladech shodu znaků (*Copy*).

Tabulka 3.4 *Přiřazení dvoubitového binárního kódu editačním operacím*

editační operace	*kód*	*následuje*
R replace / náhrada	00	pozice a báze
I insert / inzerce	01	pozice a báze
D delete / delece	11	pozice

Uvažujme dva řetězce q a p. Editační předpis $\lambda(q,p)$ vyjadřuje seznam operací pro transformaci q do p.

Původní část sekvence budu dále označovat jako q, část nepřesného opakování jako p.

Tabulka 3.5 *Editační předpis, příklad A*

původní část	g	a	c	c	g	t	c	a	t	t
nepřesné opakování	g	a	c	c	t	t	c	a	t	t
operace	C	C	C	C	R	C	C	C	C	C

Tabulka 3.6 *Editační předpis, příklad B*

původní část	g	a	C	c	g	t		c	a	t	t
nepřesné opakování	g	a	C	c		t	t	c	a	t	t
operace	C	C	C	C	D	C	I	C	C	C	C

Existuje nespočet postupů, jak pomocí uvedených operací přepsat jeden řetězec do podoby druhého.

Editační předpis pro příklad A (Tabulka 3.5) je $\lambda(gaccgtcatt, gaccttcatt) = (R, 4, t)$.

Pro příklad B (Tabulka 3.6) je $\lambda(gaccgtcatt, gaccttcatt) = (D, 4), (I, 5, t)$.

Pomocí Dvoubitového kódování lze zapsat sekvenci *gaccttcatt* (nepřesné opakování) pomocí 20 bitů. Aplikujeme přiřazení kódu z kapitoly 3.1.

`gaccttcatt` – `10 00 01 01 11 11 01 00 11 11` *(20 bitů)*

Je více metod jak zakódovat nepřesné opakování vzhledem k původní části:

- *Metoda přesné shody (exact matching method)* - dvojice čísel (pozice, délka) vyjadřuje přesnou shodu části q s p. Číslo kódujeme 4 bity, bázi/znak 2 bity. Jeden bit indikuje, jestli je následující část opakování (0 - číselná dvojice) nebo báze (1).

`(0, 4), t, (5, 5)` – `0 0000 0100 1 11 0 0101 0101` *(21 bitů)*

Tabulka 3.7 *Ukázka čtení kódu - metoda přesné shody*

hodnota	*Instrukce*
	přečti 1 bit
0	opakování, přečti 2 čísla ve 2x 4 bitech
0000	od pozice 0
0100	v délce 4 znaků
	přečti 1 bit
1	báze, přečti bázi ve 2 bitech
11	báze t
	přečti 1 bit
0	opakování, přečti 2 čísla ve 2x 4 bitech
0101	od pozice 5
0101	v délce 5 znaků

- Metoda přibližné shody (*approximate matching method*) – v této variantě je opět 1 bit pro indikaci, zda následující část je opakování (0 - číselná dvojice) nebo editační předpis (1). Kódy pro předpis jsou uvedeny výše (Tabulka 3.4). Číslo kódujeme 4 bity, bázi/znak 2 bity.

Příklad A:

`(0, 10), (R, 4, t)` – `0 0000 1010 1 00 0100 11` *(18 bitů)*

Tabulka 3.8 *Ukázka čtení kódu - metoda přibližné shody, příklad A*

hodnota	*Instrukce*
	přečti 1 bit
0	opakování, přečti 2 čísla ve 2x 4 bitech
0000	od pozice 0
1010	v délce 10 znaků
	přečti 1 bit
1	editační předpis, přečti 2 bity
00	replace / náhrada, přečti pozici ve 4 bitech a bázi ve 2 bitech
0101	na 4. Pozici
11	báze t

Příklad B:

```
(0, 10), (D, 4), (I, 4, t) – 0 0000 1010 1 11 0100 1 01 0100 11
(25 bitů)
```

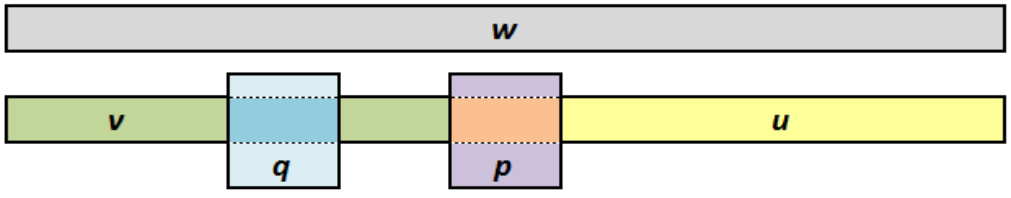
Tabulka 3.9 *Ukázka čtení kódu - metoda přibližné shody, příklad B*

hodnota	*Instrukce*
	přečti 1 bit
0	opakování, přečti 2 čísla ve 2x 4 bitech
0000	od pozice 0
1010	v délce 10 znaků
	přečti 1 bit
1	editační předpis, přečti 2 bity
11	delete / delece, přečti pozici ve 4 bitech
0100	na 4. Pozici
	přečti 1 bit
1	editační předpis, přečti 2 bity
01	insert / inzerce, přečti pozici ve 4 bitech a bázi ve 2 bitech
0100	na 4. Pozici
11	báze t

Pro tuto ukázku jsme v případě metody přibližné shody pro příklad A dosáhli nejkratšího kódu. Zkrátili jsme zápis o 2 bity na celkovou délku 18 bitů. Oproti původnímu dvoubitovému kódování v délce 20 bitů dosáhly ostatní metody zápisu nárůstu 1 a 5 bitů.

Kódováním pomocí 4 bitů můžeme dosáhnout pouze 16 různých čísel (2^4, hodnoty $0 - 15$). Tímto jsme omezeni v adresování pozic na krátké sekvence.

GenCompress je založený na slovníkové metodě komprese. Na vstupu je sekvence označená w. Uvažujme situaci, kdy už je zakódována část vstupu v a zbývá část u, zapsáno rovnicí jako $w = vu$. Algoritmus hledá v části u nejvhodnější prefix p, který se přibližně shoduje s řetězcem q z části v a zároveň ho lze (včetně editačních předpisů) vhodně přepsat do kratšího kódu. Po zakódování prefixu p se připojí tato část k v a odebere z u. Pokud se nepodaří najít přibližnou shodu prefixu p v části v, kterou lze úsporně zakódovat, odebere první znak (bázi) z části u, zakóduje ho pomocí Aritmetického kódování druhého řádu a připojí k části v. Tento postup se opakuje, dokud není část u prázdná.



Obrázek 3.6 Znázornění zakódované části v a neprocesované části u, prefix p se přibližně shoduje s částí q

Vhodnost prefixu p je rozhodována podle podmínky C, která je definována dvěma konstantami. První konstanta je k a vyjadřuje délku části sekvence. Druhá konstanta b znamená počet editačních operací.

$$C = (k, b)\ ,\ |q| > k\ ,\ |\lambda(q,p)| \leq b \qquad (\mathbf{3}.1)$$

Podmínka C je splněna, pokud je délka přibližně shodného řetězce q větší než zadané číslo k a zároveň je počet editačních operací pro přepis q do p menší nebo roven konstantě b.

Autoři experimentovali s nastavením hodnot této podmínky a došli k číslům $k = 12$ a $b = 3$, pro která dosahovali nejlepších výsledků. Tyto hodnoty poté používali ve všech svých testech.

$$C = (12,3)$$

Dalším měřítkem rozhodnutí, jestli je prefix p vhodné kódovat vůči části q za pomocí editačního předpisu λ, je *kompresní zisk (Compression gain)*. Ten vyjadřuje počet bitů, které zakódováním můžeme ušetřit.

Vzorec funkce pro výpočet kompresního zisku:

$$G(q,p,\lambda) = max\{2|p| - |(i,|q|)| - w_\lambda \cdot |\lambda(q,p)| - c, 0\} \qquad (\mathbf{3}.2)$$

q	část řetězce již zakódované části
p	prefix nekódované části
λ	editační předpis
i	pozice, na které se vyskytuje q
$2\|p\|$	počet bitů, které by byly nutné pro dvoubitové kódování p
$\|(i,\|q\|)\|$	délka kódované dvojice (pozice i a délka q)
w_λ	průměrný počet bitů pro zakódování editačních operací
$\|\lambda(q,p)\|$	počet editačních operací v $\lambda(q,p)$
c	horní strop délky řídících bitů

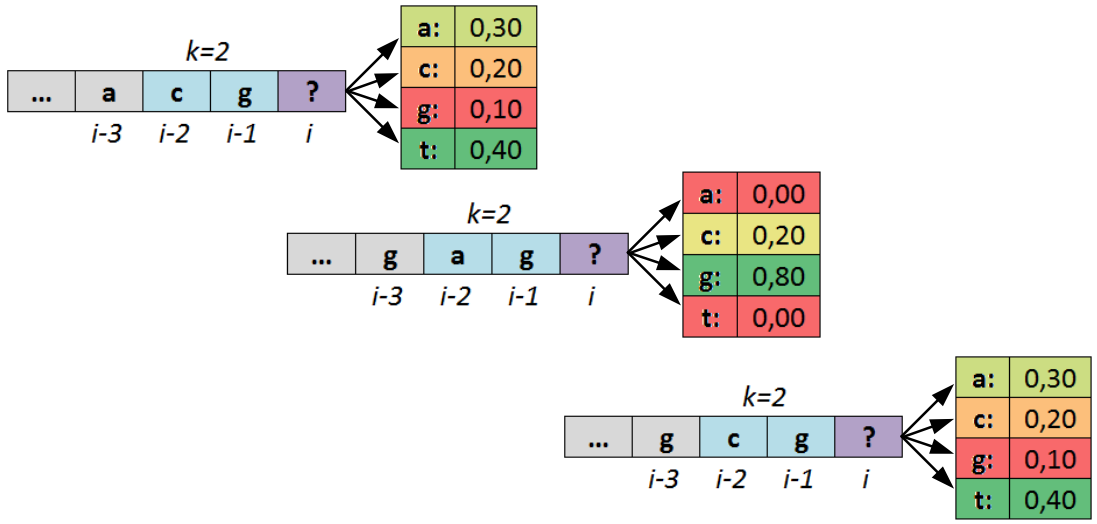
Pro vyhledávání shody prefixu algoritmus nepoužívá sufixový strom, který je dosti náročný na paměť. Místo toho hledá shodu opakování o kratší délce. Až pokud najde shodu kratšího řetězce, zkouší porovnat znaky v plné délce.

Jednoprůchodový algoritmus programu GenCompress vyhledává kromě přibližného opakování i přibližné komplementární palindromy. Postup a pravidla jsou stejná, až na rozdíl v použití bázově doplňkového reverzního řetězce části *v* jako další databáze pro vyhledávání komplementárního palindromu.

3.5 CDNA

Program *CDNA* od Davida Loewensterna a Petera Yianilose z roku 1999 je první čistě statistickým algoritmem [14].

Využívá *Markovův model*, který je definován množinou pravděpodobností přechodů z nějakého výchozího stavu (nebo ze sekvence předchozích stavů zadané délky) do stavu následujícího na základě již zpracovaných dat. U těchto modelů se bavíme o velikosti kontextu a *k-tém* řádu. Velikost kontextu vyjadřuje jak stará historie je brána v potaz při výpočtu pravděpodobností. Pokud není velikost kontextu přímo uvedena, uvažujeme o celé délce vstupních dat. Řád určuje, na kolika předchozích stavech závisí pravděpodobnost následujícího stavu.



Obrázek 3.7 Ukázka Markovova modelu druhého řádu. Pro symbol na pozici i je předpovězena pravděpodobnost na základě $k = 2$ předchozích symbolů. První a třetí případ má stejné odhady, neboť 2 předchozí symboly jsou shodné.

Při velikosti množiny možných symbolů n je pro model k-tého řádu nutné určit n^{k+1} hodnot. Pro DNA ($\{a, c, g, t\}: n = 4$) a model druhého řádu ($k = 2$) je to 64 (4^{2+1}) podmíněných pravděpodobností: $\{P(a|aa),\ P(a|ac),\ P(a|ag),\ P(a|at), \dots,\ P(t|tt)\}$.

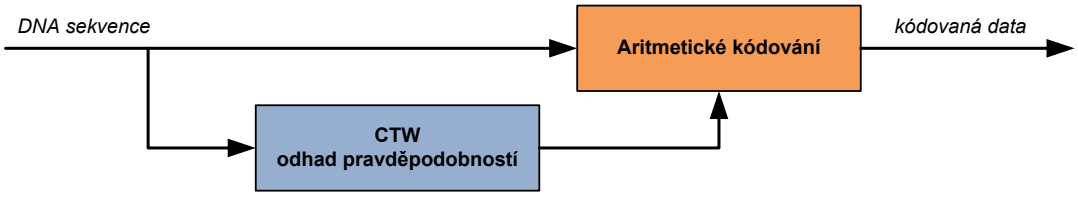
Konkrétně u CDNA je předpovídáno pravděpodobnostní rozložení následujícího symbolu pomocí kombinace *relaxovaných Markovových modelů* vyšších řádů. Jde o takový Markovův model, který za kontext nepovažuje pouze sekvence přesně shodné s kontextem dané pozice, ale i jiné velmi podobné (přibližné) sekvence. Podobnost je měřena *Hammingovou vzdáleností*. Ta udává počet záměn nutných k přepisu z jednoho řetězce na druhý. Tímto způsobem získává model mnoho předpovědí pro následující symbol. Přesnější modely předpovědí jsou oceněny větší váhou. Pravděpodobnosti jsou poté kombinovány na základě těchto vah.

3.6 CTW+LZ

V roce 2000 byla vydána publikace od japonských tvůrců T. Matsumoto, K. Sadakane a H. Imai [15]. Popisuje další program pro kompresi DNA i proteinových sekvencí.

Používá patentovaný algoritmus *Context-Tree Weighting* (*CTW*) [16], který se ukázal být schopným univerzálním nástrojem pro kompresi dat. Sám o sobě dosahuje při kódování

biologických sekvencí kompresního poměru pod 2 bity na bázi. Je postaven na binárním stromu a odhadování podmíněné pravděpodobnosti pomocí mnoha modelů o variabilní délce kontextu. Strom je budován dynamicky a obsahuje všechny zpracované podřetězce o zadané délce. Každá cesta v něm má svou váženou pravděpodobnost. Odhadnutá distribuce může být použita v Aritmetickém kodéru (a dekodéru) pro efektivní kódování následujícího symbolu.



Obrázek 3.8 CTW vytváří model pravděpodobností pro Aritmetické kódování

Japonský tým se pokusil tyto výsledky ještě vylepšit. Svou metodu přizpůsobili charakteristickým vlastnostem DNA, tedy obsahovaným repeticím. Pro jejich rozpoznávání používají *non-greedy* algoritmus s vyhodnocováním. Ten je sice pomalejší než hladový algoritmus, ale umožňuje najít delší úseky opakování. Vyhledávání je zaměřeno na přímá a invertovaná opakování (*reverse complement*s), obojí v přesné i přibližné podobě. Na ně je aplikována slovníková metoda typu LZ77 s použitím aritmetického kódování pro editační operace. Pro části, které nejsou obsaženy v repeticích, je použito CTW o řádu 32. Hloubka kontextu 32 předchozích symbolů se při experimentech ukázala být optimální v mnoha případech, z tohoto důvodu ji autoři zachovali ve své implementaci.

Kombinací algoritmů CTW+LZ kompresní poměr vylepšili a překonali výsledky předchozích metod. Avšak celkově je doba běhu tohoto programu velmi dlouhá. Jedna z kratších DNA sekvencí o délce 40 000 znaků potřebuje zhruba 8 minut. U delší sekvence obsahující 230 000 bází běžel program několik hodin.

3.7 DNACompress

Tvůrci programu GenCompress X. Chen a M. Li pokračovali ještě dále ve vylepšování svých postupů. Dali se do týmu s B. Ma a J. Trompem a v roce 2002 vydali *DNACompress* [18].

Zásadní změnou je postup při vyhledávání přibližných opakování (a přibližných komplementárních palindromů). Metoda „hladového“ vyhledávání, kterou používá GenCompress, nemusí zachytit delší opakování a je velmi časově náročná. Proto tuto část nahradili novým nástrojem *PatternHunter* z vlastní tvorby [19].

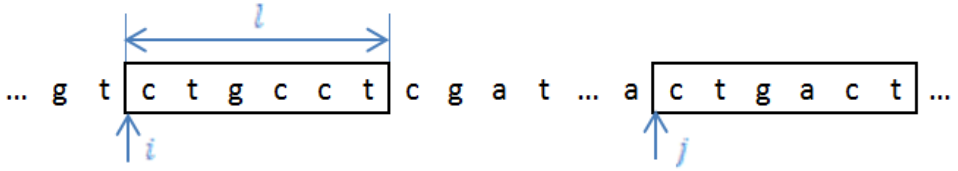
PatternHunter je samostatný program, který prochází DNA sekvenci a vyhledává v ní homologie (přibližná opakování a přibližné komplementární palindromy). Není to jediný program tohoto zaměření. Podobné sekvence vyhledává např.: FASTA, SIM, Blast, SEN-SEI a další. PatternHunter vykazuje oproti jiným programům lepší citlivost při vyhledávání, vyšší rychlost a menší nároky na paměť. Motto vyhledávacího algoritmu je „očekávej méně, abys

získal více“. Např. při nastavení vyhledávání shody 11 znaků v sekvenci používá model "111010010100110111", kdy 1 prezentuje nutnou shodu znaků mezi řetězci a 0 znamená jen možnou shodu. Při nálezu takto podobných opakování se program snaží o rozšíření shody dále do šířky. Výstupem je textový soubor, který obsahuje seznam všech nalezených řetězců seřazených od nejvyššího skóre (nejlepší shoda). Tento seznam je spolu s DNA souborem vstupem pro DNACompress, který kóduje sekvenci vůči nalezeným opakováním.

Kódování opakujících se částí vychází z metodiky GenCompress a je následující:

- Příznak, zdali jde o opakování nebo komplementární palindrom.
- Trojice (l, i, j), která definuje opakování pomocí délky l, startovní pozice předchozího výskytu řetězce i a startovní pozice následujícího výskytu řetězce j (Obrázek 3.9).
- Celkový počet editačních operací v předpisu pro nepřesné opakování.
- Všechny editační operace zapsané v trojici (e, o, b), kde e vyjadřuje editační operaci (*replace*, *insert*), o je pozice (*offset*) rozdílu a b představuje bázi. V případě delece jde pouze o dvojici (e, o). Místo kódování každé operace zvlášť lze seřadit posloupné operace stejného druhu do bloků a kódovat jen pozice a báze, což může vést ke kratšímu kódu.

Tak jako GenCompress, používá DNACompress podobné podmínky a funkce pro rozhodnutí, kdy je nalezený řetězec vhodné kódovat jako opakování. Na nevyhovující řetězce a zbylé části, je opět aplikováno Aritmetické kódování druhého řádu.



Obrázek 3.9 Ukázka výskytu přibližného opakování zapsaného trojicí (l, i, j)

3.8 NML, GeNML

Ioan Tabus a jeho kolegové zveřejnili v roce 2003 svou kompresní metodu nazvanou *Normalized Maximum Likelihood* (*NML*) [20].

NML rozděluje DNA sekvenci do bloků o pevné délce. K zakódování bloku algoritmus najde podřetězec (přímou shodu nebo palindrom), který se předtím vyskytl na vstupu a má od něj minimální Hammingovou vzdálenost. V úvahu jsou brány pouze substituční (replace) operace. Tento předchozí výskyt nemusí být nutně na začátku některého z bloků. Zpracovávaný blok je kódován jako reference na tento předchozí výskyt s použitím bitové masky pro reprezentaci změn. Bitová maska je interpretována Aritmetickým kódováním (nultého řádu). V případě, kdy tento způsob použití pointeru a masku nedosahuje dostatečné efektivity, je aplikována optimálnější metoda z výběru: Markovův model prvního řádu s Aritmetickým kódováním nebo Dvoubitové kódování. Pro interpretaci čísel je použitá pevná délka kódu,

protože jsou častěji kódovány čísla, jejichž interpretace pomocí variabilní délky kódu není přínosná.

Dobré výsledky metody NML vedly autory k další optimalizaci jejich přístupu. Po dvou letech byl vydán *GeNML* [21] a následně ještě vylepšen [22]. Mezi rozdíly je výběr fixní délky bloku (24, 32, 40 nebo 48), kódování bitové masky pomocí vlastního NML modelu a hlavně komplexnější kombinace modelů vyšších řádů.

Tato skupina programů se řadí mezi první, které lze prakticky aplikovat i na větší biologické sekvence vzhledem k dosahovaným výsledkům a době běhu komprese/dekomprese. Dokonce i nasazení dalších pravděpodobnostních modelů v metodě GeNML, které vedly ke zlepšení kompresních poměrů, zachovalo časovou náročnost v použitelné míře.

3.9 DNAC

V roce 2004 implementoval Chun-Ho Chang kompresní program *DNAC* v jazyce Java [23]. Spojuje v něm úspěšně postupy několika předchozích nástrojů (zejména Cfact a GenCompress). Dosáhl k celkovému zlepšení z pohledu kompresní účinnosti i doby běhu komprese.

Algoritmus se skládá ze 4 fází:

1. Vytvoření sufixové stromu k vyhledání přesných repetic
2. Přesná opakování jsou dále rozšířena na nepřesná opakování s použitím editačních operací a podle vyhodnocení bitového zisku
3. Výběr optimálních nepřekrývajících se opakování
4. Kódování čísel pomocí Fibonacciho řady a dvoubitové kódování na zbylé sekvence

V této práci Chang pracuje pouze s přímým opakováním, nezabývá se zrcadlovými výskyty, komplementy a komplementárními palindromy.

3.10 DNAPack

DNAPack pochází z roku 2005 od dvojice B. Behzadi a F. Le Fessant [24]. Byl napsaný v jazyce OCaml.

Tento program se opět vyhnul hladovému algoritmu pro vyhledávání repetic a nasazuje dynamické programování. Podobnost přibližných řetězců je určována Hammingovou vzdáleností. DNAPack používá z editačních operací pouze substituci. Inzerci a Deleci lze zařadit do jejich algoritmu, avšak autoři tyto operace vynechali z důvodu snížení časové náročnosti a komplexity.

Části sekvence, které nejsou identifikované jako repetice, jsou kódovány nejlepší volbou z Aritmetického kódování druhého řádu, CTW nebo dvoubitové kódování. Všechna čísla a pozice jsou interpretována Fibonacci kódováním.

Struktura komprimovaného souboru má 3 oblasti:

- HEADER – hlavička obsahující všechny nutné informace pro dekódování CODE a BASES, např.: typ použité komprese, počet segmentů v CODE, minimální délku shodného prefixu repetic, nejčastější bázi používanou při substituci za jinou bázi v repeticích.

- CODE – kódovací oblast se skládá ze dvou různých typů. Repetice a zbylé části. Pro zbylé části obsahuje pouze délky segmentů. U repetic obsahuje jednobitový příznak, zdali jde o přímé nebo invertované opakování. Dále vzdálenost k původnímu výskytu a soubor operací (kopie, mutace) k rekonstrukci konkrétní repetice.

- BASES – série segmentů popsaných v CODE oblasti, které jsou dohromady zakódované nejvýhodnějším algoritmem uvedeným v HEADER.

3.11 XM

M. Cao a jeho tým představili v roce 2007 program *Expert Model* (*XM)* implementovaný v jazyce Java [25]. Postup, který tento program aplikuje, je vysoce účinný pro kódování DNA i proteinových sekvencí.

Hlavní myšlenka spočívá v tom, že jakékoliv statistické modely, které předpovídají pravděpodobnostní distribuci pro následující znak, je možno zkombinovat. Jednotlivé předpovědi jsou v průběhu kódování sekvence různě úspěšné. To algoritmus zpětně kalkuluje a v každém kroku znovu vyhodnocuje přesnost daného modelu, čímž udává jeho aktuální váhu. Pro kombinování vážených pravděpodobností vychází autoři z *Bayesovy věty* o podmíněné pravděpodobnosti. Poté, co dojde k výpočtu kombinovaných pravděpodobností, je symbol zakódován primárním kompresním algoritmem, kterým je Aritmetické kódování.

Algoritmus pracuje s těmito statistickými modely (experty):

- *Copy expert* – Tento model je zaměřen na opakované části (přímé shody). Rozmýšlí se nad následujícím symbolem, jestli není součástí kopie některého z předchozích podřetězců se vzdáleností f. Jinak řečeno symbol na pozici i by mohl být s jistou pravděpodobností stejný jako symbol na pozici $i - f$. Pravděpodobnost P je počítána vzorcem následujícím vzorcem.

$$P = \frac{r+1}{w+2} \qquad (\mathbf{3}.3)$$

  Veličina r představuje počet správných predikcí, které expert provedl a w vyjadřuje délku okna, se kterou expert pracuje. Zbylá pravděpodobnost $(1 - P)$ je rovnoměrně přidělena zbylým symbolům z abecedy.

- *Reverse expert* – Toto je ekvivalent Copy experta pro reverzní komplementy opakovaných řetězců.

- *Markov expert* – Markovův model druhého řádu pro DNA a prvního řádu proteinové sekvence

- *Context Markov expert* – Další Markovův model, tentokrát není distribuce pravděpodobností založena na celé historii výskytu symbolů v sekvenci, ale pouze na pevně limitovaném kontextu. Konkrétně je nasazen model prvního řádu s kontextem o délce 512 předchozích symbolů.

4 Vlastní metoda

Při pohledu na předchozí kapitolu a existující programy, které fungují na základě kódování jedné sekvence, je zřejmá společná myšlenka. Většina autorů sdílela stejnou představu o ideální metodě pro kompresi biologické sekvence. Jedná se o využití nejdelších (přibližných) repetic, zaznamenání jejich dalších výskytů za pomoci minimálního nutného počtu řídících bitů a poté aplikování entropického kódování (např. Aritmetické kódování 2. řádu). Tímto způsobem kombinují kategorie kódovacích algoritmů – zápis vícenásobných výskytů pomocí indexu lze považovat za slovníkovou metodu, kódování čísel a zbylých částí představuje statistické metody. Použité dílčí algoritmy se mohou lišit, ale hlavní idea zůstává stejná. Nalezení a výběr opakujících se částí se zdá být při tomto postupu jako klíčové. Podpůrné funkce, které dopředu vypočítávají počet bitů pro jejich zápis, vyhodnocují zisk při použití daných repetic a pomáhají při jejich výběru. Může však docházet k rozporu, kdy lokální zisk nemusí souhlasit s globálním vylepšením úspory bitů.

4.1 Faktor 2bitového kódování

Možnost použití 2 bitů na zápis jedné báze je jeden z hlavních faktorů, který činí kódování DNA výzvou v oblasti komprese dat. Autoři čelí při návrhu metod interpretace často rozporu, kdy pro vložení své logiky musí obětovat část zápisu řídícím bitům.

V biologických sekvencích jsou poměrně často vidět úseky jako *aaaaaaaaaa* nebo *gtgtgtgtgtgtgt*. Je velice lákavé zaměřit se na tyto sekvence a kódovat je některým ze způsobů *Run-Length Encoding* (*RLE*). Při jejich nasazení by uvedené příklady mohly vypadat např. jako *a9* (zápis opakující se báze a počet dalších opakování) a *2gt6* (počet bází v opakování a počet dalších opakování). Jenže při interpretaci na úrovni binárního kódu, kdy jsou k dispozici jen 2 znaky (*0,1*), je nutné znakům přiřadit větší počet bitů, než je pro zápis jedné báze nutný. Při použití 2 bitů na symbol je k dispozici rozdílný kód pouze pro 4 symboly (*00*, *01*, *10*, *11*), které jsou spotřebovány pro 4 báze DNA. Rozšířením abecedy kódu o další znaky, které by např. vyjadřovaly čísla (počet opakování), začne délka kódů značně narůstat, pokud není obsah sekvence dostatečně homogenní jako třeba v případě několikanásobného výskytu jednoho či dvou bázi.

Jednou z možností je rozdělit biologické sekvence do bloků o pevné délce, kde by mohl první bit v bloku avizovat, zdali jde o klasický 2bitový zápis nebo jiný způsob interpretace. Opět ale narážíme na problém, kdy kvůli různorodosti DNA řetězce nebývá zisk ze zkráceného zápisu dostatečně vyvážen potřebou vložení řídících bitů.

Další cestou je vložení čísla na začátek každého sektoru, které by udávalo jeho délku. Tento sektor by byl celý zapsán v binárním kódování nebo ve vhodnějším kódu. Algoritmus je obeznámen s tím, že po přečtení celého sektoru bude opět následovat nejdříve číslo a až po něm vlastní kód. Čísla mohou být zapsány v prefixovém kódu a tímto by se program vyhnul zápisu nevyužitých bitů při použití pevné délky. Za tímto číslem by následoval řídící bit, který by udával typ reprezentace následujícího kódu.

Jiná myšlenka tkví v použití sudých a lichých kódů. Na začátku každého bloku by opět bylo nejdříve číslo, které by vyjadřovalo jeho délku. Poté by se nemusel vyskytovat řídící bit, ale rovnou kódovaná sekvence ve zvolené interpretaci. V případě 2bitového kódování je zřejmé, že délka jeho kódu bude mít vždy sudý počet. Lichý počet, o různé délce (např. 3, 5, 7, 9 nebo třeba rozlišen zbytkem po celočíselném dělení délky), by mohl znamenat i několik odlišných způsobů kódování.

Délka bloku nemusí být nutně vyjádřena přímou interpretací čísla. Může být výhodnější nasazení relativních délek vzhledem k předchozímu bloku. Při mých experimentech jsem dosahoval zkrácení zápisu délky následujícího bloku jiným způsobem. Nejdříve jsem interpretoval část sekvence v jejím nejvýhodnějším kódu. Poté spočítal počet výskytů jedniček a na konec kódu ještě jeden znak *1* přidal. Tento počet bitů jedna jsem použil jako zápis délky, neboť jsem věděl, že mám číst následující kód, dokud nenarazím na počet jedniček přesahující tento počet.

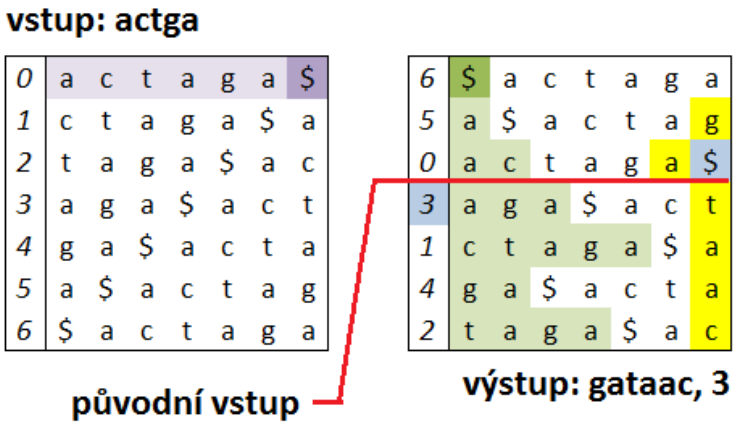
Ovšem ani s jedním z těchto uvedených přístupů jsem nebyl schopen interpretovat kód dost efektivně. Při nasazení na delší reálnou DNA sekvenci vždy počet řídících bitů převážil výhodu zkráceného kódu několika (avšak ne dostatečného počtu) sekvencí. 2bitové kódování se ukázalo být poměrně účinným způsobem kódování, které není jednoduché některým podobným přístupem překonat.

4.2 Transformace

Další ze strategií, na kterou jsem se zaměřil, bylo nasazení transformace na sekvenci ještě před aplikováním vlastního kódování. Obecně jsou myšleny takové transformace, které přeměňují data do stejně dlouhého, ale výhodnějšího zápisu. Ke zrekonstruování do zpětné podoby přitom stačí použít krátký klíč, který bývá výstupem transformačního procesu. Konkrétně jsem vyzkoušel *Burrows-Wheeler Transform* (*BWT*).

Tato metoda transformace přidá za poslední znak vstupní sekvence symbol, který označuje její konec. Poté rotuje znaky v řetězci a řadí je pod sebe, dokud nevznikne dvourozměrné pole $n \times n$, kde n představuje délku vstupu plus symbol konce. Tímto způsobem vzniknou všechny možné sufixy vstupního řetězce (od začátku až do pozice symbolu konce). Celé toto pole je poté abecedně seřazeno, kdy znak konce se řadí před ostatní znaky. Poslední sloupec tohoto pole pak představuje výstup transformace (od druhého znaku). K tomuto výstupu ještě patří index řádku, který se nachází za symbolem konce (viz. Obrázek 4.1).

Reverzním procesem je poté sada operací, kdy se výstupní sloupec abecedně seřadí a před něj opět zařadí výstupní sloupec. Seřazení tentokráte probíhá na dvojici znaků. Před seřazené dva sloupce se opět připojí výstupní sekvence. Tento postup se opakuje, dokud nedostaneme na některém z řádků stejně dlouhý řetězec, který má na konci oznamovací symbol.



Obrázek 4.1 Ukázka principu BWT na krátkém DNA řetězci

Po aplikaci této transformace by měly být symboly v řetězci „více seřazené“ a tím pádem snadněji komprimovatelné. Pouze za cenu uložení číselného klíče. Tato transformace je úspešně aplikována především na klasický text (literatura) ještě ve spojení s *Move-To-Front* a *Run-Length-Encoding*. Až poté je na data aplikován některý z kodérů.

```
before BWT:
gattcttaggtgttccttat
after BWT:
ttgttctagtttttagccga
pointer 2
```

```
before BWT:
gtgtatgtgtgtgtgcatgt
after BWT:
ctgttttttttgggggggaga
pointer 13
```

Obrázek 4.2 Dvě sekvence před a po aplikaci BWT

Aplikoval jsem tuto transformaci na různé dlouhé fixní bloky sekvence DNA. Při použití kratších bloků (délky okolo 20) se nepatrně zvýšila četnost po sobě jdoucích shodných znaků. Zároveň ale došlo k přeuspořádání dalších znaků v jiných částech kódu tak, že žádná z následně nasazených metod (Aritmetické kódování, LZ) nevedla ke zlepšení kompresních poměrů. Pro bloky o větší délce nenastala z pohledu uspořádání téměř žádná pozitivní změna.

4.3 Zvolený přístup

Opakované sekvence jsou často od sebe velmi vzdáleny. Samotné Aritmetické kódování plně nevyužije této redundance, pokud není doplněno o dostatečně velké množství kontextových modelů a efektivní vyhodnocování pravděpodobnosti. Tento způsob se zdá být účinným, avšak početně velmi náročným.

Pro překonání hranice kompresního poměru, kterou udává 2bitové kódování, jsem se rozhodnul pro využití principu založeného na slovníkové metodě v kombinaci s Adaptivním aritmetickým kódováním. Tímto způsobem chci zachytit delší opakované řetězce, které se v biologických sekvencích vyskytují. Tyto opakované výskyty nahradit indexy předchozích výskytů a zbylé části zakódovat jako celek statistickým kodérem.

Zaměřil jsem se proto na přímé nahrazení opakovaných řetězců indexy ve slovníku. Tento způsob by měl být schopen odebrat dostatečné množství nadbytečnosti ze sekvence, aniž by zakomponování kódovací logiky soubor navýšilo přes úroveň kompresního poměrů dvou bitů na jednu bázi.

Další informace ohledně aplikovaných metod jsou uvedeny v nadcházející kapitole.

4.4 Implementace

V této části představuji vlastní implementaci nástroje pro kompresi DNA s názvem *DNAcod*. Jedná se *Win32 Console Application* postavenou na platformě *.NET 4.5*. Program čte biologické sekvence zapsané v prostém formátu, přípustné jsou pouze znaky '*a*', '*c*', '*g*' a '*t*'. Aplikace pracuje ve 3 módech: komprese, dekomprese a porovnání obsahu souborů. Popis činností v jednotlivých módech je uveden v následujících podkapitolách. Při každém správném spuštění programu je vygenerován log s informacemi o běhu programu.

4.4.1 Komprese

Po načtení vstupního souboru jsou data předána třídě *Research*. Ta má za úkol identifikovat mnohačetné výskyty řetězců – repetice a jejich pozice. Dále popisuji nejdůležitější kroky této fáze. Při tomto procesu postupně vzniká několik důležitých seznamů o různých datových typech. Pro snadnější pochopení uvádím mezi jednotlivými odstavci definici těchto výstupních datových typů v pseudo-kódu.

Data jsou přečtena do seznamu překrývajících se bloků (*Suffix)* s posunem vždy pouze o jednu pozici. Bloky mají pevnou délku, výchozí nastavení je 30 znaků. Celý seznam je poté abecedně setříděn. Tento přístup je inspirován metodou BWT a umožňuje jednoduše nalézt společné podobné části podřetězců uvnitř sekvence.

```
// překrývající se bloky vstupních dat
Suffix(pozice, řetězec)
```

Následuje průchod těmito bloky, kdy jsou vždy porovnávány pouze dva po sobě jdoucí záznamy. Pro tuto dvojici se hledá nejdelší společná část řetězce, která nemusí být nutně na začátku bloku. Nalezené shody musí odpovídat podmínkám na minimální délku (*matchMinLength*, výchozí hodnota 6). Výstupem této funkce je seznam (*Match*), který obsahuje shodný řetězec a 2 pozice, na kterých byl řetězec nalezen.

```
// nalezené dvojice shod
Match(shodný řetězec, první pozice, druhá pozice)
```

Tento seznam je setříděn podle pozice prvního výskytu. Dále zpracován tak, aby sloučil překrývající se dvojice shod do delších řetězců. To znamená, že se porovnávají dvě dvojice shod, jejichž (první i druhé) výskyty jsou v průniku a okolní části sekvence jsou shodné. Před posledním využitím tohoto listu jsou shody setříděny sestupně podle délky shodného řetězce.

Nyní je na řadě početně nejnáročnější fáze, identifikace repetic (*Repeat*). Při ní je procházen list shod (*Match*) od nejdelší po nejkratší. Přitom se snaží s ostatními prvky tohoto listu sloučit výskyty. Tímto jsou v DNA sekvenci hledány nejdelší „vzory“, jejichž podřetězce se vyskytují na více místech. Následné výskyty mohou být kratší a začínat kdekoliv uprostřed vzoru. Výstupem je seznam, který obsahuje pozici vzoru, její plnou podobu a seznam výskytů (*Occurrences*). Každý výskyt (*Occurence*) obsahuje pozici vzoru, svou pozici, posun v rámci vzoru a shodnou délku se vzorem.

```
// identifikované repetice – vzory, jejichž podřetězce se
vícekrát vyskytují

Repeat(pozice, řetězec, seznam výskytů)

// výskyty

Occurence(pozice vzoru, vlastní pozice, posun v rámci vzoru,
vlastní délka)
```

V rámci datových typů Repeat a Occurence jsou některé informace ukládány zprvu nadbytečně, mají ale své využití v pozdější části kódu.

Nad identifikovanými vzory (*Repeat*), probíhají další operace, které se snaží nadbytečné záznamy o výskytech odebrat (výskyt pokryt už jiným výskytem) a překrývající se výskyty sloučit. Překrývající se vzory jsou odebrány tak, aby byly zachovány vždy delší varianty.

Hlavním výstupem v rámci třídy *Research* jsou dva seznamy. Je jím list prvních (vzorových) výskytů (*FirstOccurence*), který obsahuje pouze pozice. Označení „první“ neznamená, že by byl v rámci sekvence nejblíže počátku. Označení je myšleno tak, že je to první výskyt o největší délce (vzor), na který se odkazují následující výskyty. Následující výskyty (*NextOccurence*) jsou druhým důležitým výstupním seznamem. Ten je seřazen vzestupně podle pozice v textu, což umožňuje relativní zápis čísel a tím ušetření délky kódu v interpretaci.

```
// první výskyty

FirstOccurence(index, pozice)

// další výskyty

NextOccurence(index prvního výskytu, pozice, posun v rámci
prvního výskytu, vlastní délku)
```

Po nalezení těchto výskytů následuje vlastní kódování, které řídí třída *Algorithm*. Tato třída projde seznam dalších výskytů (*NextOccurence*) a podřetězce, které tyto výskyty pokrývají, jsou následně odebrány z kódované DNA sekvence. Tento úkon musí probíhat v pořadí od nejvzdálenějších výskytů, aby nedocházelo k rozporu pozic u dřívějších výskytů. Ve zbytku DNA sekvence tedy zůstanou všechny první výskyty (*FirstOccurence*) a ty části, které nejsou repeticemi pokryty.

V této fázi je možno začít s tvorbou výstupního kódovaného souboru. Všechna čísla jsou interpretována pomocí Fibonacciho kódování (bez posunu). Na začátek souboru je zapsána trojice čísel:

- počet prvních výskytů (*FirstOccurence*)
- počet dalších výskytů (*NextOccurence*)
- délka zbylého kódu DNA sekvence

Následuje zápis relativních pozic prvních výskytů (*FirstOccurence*). Pořadí, v jakém je ve výstupním souboru první výskyt (*FirstOccurence*) zapsán, představuje jeho index.

Další je připojen výčet dalších výskytů (*NextOccurence*). Každý záznam obsahuje čtveřici čísel:

- index prvního výskytu (*FirstOccurence*)
- relativní pozice (od předchozího záznamu)
- posun v rámci prvního výskytu (*FirstOccurence*)
- délku výskytu

Index je kvůli použité interpretaci čísel ve Fibonacci kódu navýšen o jeden. Pokud je posun v rámci prvního výskytu nulový (to znamená, že další výskyt začíná na první pozici prvního výskytu), je zapsán bit *0*. V případě, kdy posun není nulový, je zapsán bit *1* a až poté následuje kód pro dané číslo. Při mých zkušebních testech, převládal nulový posun. Při nasazení Fibonacciho kódování bych musel pro zápis nuly opět číslo navýšit o jedna (*0→1, 1→2, ...*). Proto jsem se rozhodnul o změnu interpretace.

Poté, co jsou zapsány všechny další výskyty (*NextOccurence*), pokračuje zápis zbylé DNA sekvence. Ta je kódována pomocí Adaptivního aritmetického kódování. V případě kódování biologické sekvence očekávám pouze 4 možné symboly. Na začátku komprese jsou všechny četnosti nastaveny na stejnou hodnotu (1), ale v průběhu zpracovávání vstupních dat se četnosti přizpůsobují, což vede ke změně v rozložení intervalů. Zvolil jsem implementaci bez zakódování pátého znaku, který by avizoval konec kódované zprávy. Místo toho uchovávám zvlášť informaci o původní délce kódu, kterou při zpětném procesu dekomprese předávám dekodéru spolu se vstupními daty. Bez této informace by dekodér dále procházel intervaly a generoval chybný výstupní kód.

Celý obsah je zapsán binárně do výstupního souboru, čímž proces komprese končí. Jako vedlejší produkt při implementaci a ladění kódu jsem přidal funkci, která zapisuje do zvláštního souboru výsledný kód v binární podobě spolu s jeho významem v lidsky čitelné podobě. Následující obrázky (Obrázek 4.3, Obrázek 4.4, Obrázek 4.5) obsahují úryvky z tohoto souboru pro vybranou DNA sekvenci (*vaccg.dat*). Na přiloženém CD je v přílohách jeho celá podoba (*vaccg.view*).

```
0100100101011 0000000011 10100010101000010101001011| firstOcc=366  nextOcc=firstOcc+55=421 restlength=184709
00011 | (0) rpos=5/apos=5
10100101000011 | (1) rpos=428/apos=433
010101001010011 | (2) rpos=829/apos=1262
00000100000010011 | (3) rpos=1987/apos=3249
10100100000011 | (4) rpos=394/apos=3643
0001001001001011 | (5) rpos=1479/apos=5122
00001000101000011 | (6) rpos=1804/apos=6926
1010010010011 | (7) rpos=305/apos=7231
```

Obrázek 4.3 Ukázka interpretace kódu a významu – úvodní počty a první výskyty

```
01001011 0000100000101011 1 100011 0000011 | indF=30+1 rpos=1516/apos=10811 from=9 len=13 '---------tatttttatatta-----
100011 00101010000011 1 10011 0000011 | indF=8+1 rpos=409/apos=11220 from=6 len=13 '------tttttcataatta--' indR=21
10010011 0010010001010011 0 0010011 | indF=26+1 rpos=1325/apos=12545 from=0 len=16 'ttagtttatactatta--------' indR=56
0100000010011 00100101010011 1 11 0000011 | indF=289+1 rpos=516/apos=13061 from=1 len=13 '-tttttctatgtta' indR=33
01010100011 00001000100011 0 1000011 | indF=108+1 rpos=440/apos=13501 from=0 len=14 'cgtatattttattt----' indR=88
01011 101000000011 0 0000011 | indF=6+1 rpos=148/apos=13649 from=0 len=13 'agataaagtagat---' indR=166
0100100010011 1000010001011 0 1010011 | indF=297+1 rpos=336/apos=13985 from=0 len=17 'atttattttattctatt-----' indR=55
0100100001011 01010010011 0 1000011 | indF=331+1 rpos=117/apos=14102 from=0 len=14 'aattaataatataa---' indR=25
01001000011 0000000100001011 0 1000011 | indF=98+1 rpos=1398/apos=15500 from=0 len=14 'ttattatcatcatt---' indR=89
0010010010011 010010001011 0 1000011 | indF=303+1 rpos=209/apos=15709 from=0 len=14 'aaacagtttaatag-----' indR=90
100100001011 1001000000011 0 0000011 | indF=204+1 rpos=239/apos=15948 from=0 len=13 'ttataacatcgta-' indR=185
```

Obrázek 4.4 Ukázka interpretace kódu a významu – další výskyty

```
gagaataaaaaatattttagtgagaccatcgaagagagaaagagataaaacttttttacgactccatcagaaagaggt
gagaata                3gaccatcga                                2atcagaaagaggt
```

Obrázek 4.5 Ukázka vypuštění repetic, index na první výskyt je vložen jen pro účely zobrazení

4.4.2 **Dekomprese**

Proces dekomprese je podstatně jednodušší. Její chod opět řídí třída *Algorithm*.

Po načtení vstupního souboru do paměti probíhá nejdříve dekódování 3 čísel, která udávají počet záznamů ve slovnících (prvních a dalších výskytů) a délky zbytku sekvence.

Následuje načtení a dekódovaní obou slovníků.

Adaptivní aritmetický dekodér interpretuje zbytek kódu v rámci uvedené délky.

Nejzajímavější fáze je procházení seznamem dalších výskytů, při kterém jsou (přes index na první výskyty) zjištěny pozice vzorů, ze kterých je rekonstruován řetězec dalšího výskytu. Ten je vložen na svou pozici a tímto dochází k expanzi zbytku kódu do původní podoby. V tomto kroku musí být správně přepočtena pozice výskytu, neboť vzory se vyskytují na pozicích před i po dalších výskytech. Když je sekvence DNA zredukována o další výskyty, přestanou odpovídat pozice prvních výskytů. A následným každým vložením před jejich pozici se jejich umístění ještě mění.

4.4.3 **Porovnání obsahu**

Tento mód programu DNAcod slouží pro rychlou detekci chyb mezi originálním souborem a jeho kopií. Sekvence jsou porovnávány znak po znaku a každá odchylka je zobrazena s číslem pozice.

4.4.4 Zdrojový kód

Zdrojový kód celého projektu je umístěn na přiloženém CD. Zde je pouze stručně uveden seznam vybraných tříd a jejich význam:

Program.cs – Startovací třída, která přebírá vstupní parametry z příkazového řádku a poté spouští hlavní řídící program.

Algorithm.cs – Hlavní řídící kód. Obsahuje nadřazené funkce pro všechny módy běhu programu: kompresi, dekompresi, porovnání souborů.

Research.cs – Stěžejní třída pro mód komprese. Je zaměřená na hledání opakovaných výskytů. Obsahuje struktury *Suffix, Match, Interval*.

Repeat.cs – Další třída určená pro fázi komprese. Obsahuje kód pro opakované vzory a výskyty: *Repeat, Occurence, FirstOccurence, NextOccurence*.

ArithmeticCoding.cs, *ArithmeticEngine.cs*, *ArithmeticDNA.cs* – Skupina souborů, která slouží pro Adaptivní aritmetické kódování. Postupně je jejich význam následující: zapouzdření metod komprese a dekomprese; vlastní kódování založené na přidělování intervalů podle pravděpodobností symbolů; model upravený pro DNA sekvenci.

Fibonacci.cs – Kód pro interpretaci čísel ve Fibonacciho kódu. Rovněž obsahuje metody pro k-Shifted Fibonacci kód.

Tools.cs – Pomocné statické funkce, především pro práci se soubory.

Log.cs – Práce s výpisem na obrazovku a zápis protokolu.

4.4.5 Poznámky

Jako základ pro zdrojový kód Adaptivního aritmetického kódování jsem použil projekt, který je vedený pod licencí The Code Project Open License (CPOL) 1.02[4] [26].

Funkce pro jednoduché Fibbonaciho kódování čísel vychází z ukázkových řešení předmětu Komprese dat[27] od pana doc. Ing. Jana Platoše, Ph.D.

Syntaxe a příklady použití programu:

- Komprese - "DNAcod.exe c [input file] [output file]"
  DNAcod.exe c humdyst.dat humdyst.cod
- Dekomprese - "DNAcod.exe d [input file] [output file]"
  DNAcod.exe d humdyst.cod humdyst.dec
- Porovnání souborů - "DNAcod.exe x [orig file] [copy file]"
  DNAcod.exe x humdyst.dat humdyst.dec

[4] http://www.codeproject.com/info/cpol10.aspx

5 Porovnání výsledků

Porovnávání výsledků kompresních nástrojů pro DNA sekvence je možné pouze v omezené míře. Často není k dispozici spustitelný program nebo celý zdrojový kód. Některé (většinou starší) metody nelze použít na data větších velikostí, kvůli jejich náročnosti na pamět a čas. Další nástroj DNACompress (popsaný v kapitole 3.7), používá jako první fázi vyhledávací aplikaci PatternHunter, která není volně dostupná. Jsme v podstatě odkázáni na výsledky, které autoři uvedli ve svých publikacích.

V průběhu let se ale ustálila sada souborů s DNA v jednoduchém textovém formátu (viz. kapitola 1.5.3), na kterou jsou často kompresní programy testovány a dosažené výsledky jsou zveřejněny. Jedná se o skupinu 9 sekvencí, kde 2 obsahují genom chloroplastu (*CHMPXX, CHNTXX*), 4 jsou lidské (*HUMDYST, HUMHBB, HUMHDAB, HUMHPRTB*), 1 mitochondrie (*MPOMTCG*) a 2 viry (*HEHCMV, VACCG*).

Tabulka 5.1 *Porovnání výsledků komprese vůči univerzálním programům (hodnoty kompresního poměru jsou uváděny v jednotkách bpb)*

Sekvence	*Délka*	*gzip(DEFLATE)*	*zip (DEFLATE)*	*7zip (LZMA:16)*	*DNAcod*
CHMPXX	121 024	2,2821	2,1600	2,0820	1,9525
CHNTXX	155 844	2,3347	2,2288	2,1624	1,9907
HEHCMV	229 354	2,3277	2,2109	2,1642	2,0147
HUMDYST	38 770	2,3627	2,3338	2,2291	1,9537
HUMHBB	73 308	2,2454	2,1893	2,0573	1,9376
HUMHDAB	58 864	2,2395	2,1644	2,0222	1,9884
HUMHPRTB	56 737	2,2667	2,1976	2,0581	1,9757
MPOMTCG	186 609	2,3290	2,2204	2,1276	1,9911
VACCG	191 737	2,2519	2,1297	2,0665	1,9513
Průměr		*2,2933*	*2,2039*	*2,1077*	*1,9728*

Ve výše uvedeném přehledu (Tabulka 5.1) je implementovaná metoda DNAcod porovnána vůči běžně používaným kompresním nástrojům. V zastoupení jsou *gzip*, *zip* a *7zip*.

První dva pracují na stejném principu, kterým je *DEFLATE*. Jde o kombinací slovníkové LZ77 metody a statistického Huffmanova kódování. Poslední dobou stále populárnější 7zip má k dispozici více metod, jak soubor zakódovat. Na tyto DNA sekvence aplikoval metodu *LZMA:16* (*Lempel-Ziv-Markov chain algorithm*), kterou předčil postup DEFLATE.

Program DNAcod, který je výsledkem této práce, dosáhnul průměrného výsledku pod 2 bity na bázi a vůči univerzálním nástrojům obstál z pohledu dosaženého kompresního poměru.

Tabulka 5.2 *Porovnání výsledků komprese vůči DNA kompresním programům, část 1 (hodnoty kompresního poměru jsou uváděny v jednotkách bpb)*

Sekvence	*Délka*	*DNAcod*	*BioComp-2*	*GenComp*	*CTW-LZ*	*DNACompr*
CHMPXX	121 024	1,9525	1,6848	1,6730	1,6690	1,6716
CHNTXX	155 844	1,9907	1,6172	1,6146	1,6120	1,6127
HEHCMV	229 354	2,0147	1,8480	1,8470	1,8414	1,8492
HUMDYST	38 770	1,9537	1,9262	1,9231	1,9175	1,9116
HUMHBB	73 308	1,9376	1,8800	1,8204	1,8082	1,7897
HUMHDAB	58 864	1,9884	1,8770	1,8192	1,8218	1,7951
HUMHPRTB	56 737	1,9757	1,9066	1,8466	1,8433	1,8165
MPOMTCG	186 609	1,9911	1,9378	1,9058	1,9000	1,8920
VACCG	191 737	1,9513	1,7614	1,7614	1,7616	1,7580
Průměr		*1,9728*	*1,8266*	*1,8012*	*1,7972*	*1,7885*

Tabulka 5.3 *Porovnání výsledků komprese vůči DNA kompresním programům, část 2 (hodnoty kompresního poměru jsou uváděny v jednotkách bpb)*

Sekvence	*Délka*	*DNAcod*	*DNAPack*	*CDNA*	*GeNML*	*XM*
CHMPXX	121 024	1,9525	1,6602	-	1,6617	1,6577
CHNTXX	155 844	1,9907	1,6103	1,6500	1,6101	1,6068
HEHCMV	229 354	2,0147	1,8346	-	1,8420	1,8426
HUMDYST	38 770	1,9537	1,9088	1,9300	1,9085	1,9031
HUMHBB	73 308	1,9376	1,7771	1,7700	-	1,7513
HUMHDAB	58 864	1,9884	1,7394	1,6700	1,7059	1,6671
HUMHPRTB	56 737	1,9757	1,7886	1,7200	1,7639	1,7361
MPOMTCG	186 609	1,9911	1,8932	1,8700	1,8822	1,8768
VACCG	191 737	1,9513	1,7583	1,8100	1,7644	1,7649
Průměr		*1,9728*	*1,7745*	*1,7743*	*1,7673*	*1,7563*

Přehledy (Tabulka 5.2, Tabulka 5.3) obsahují výsledky specializovaných programů pro kompresi biologických sekvencí. U programu CDNA nejsou k dispozici výsledky pro sekvence chmpxx a hehcmv. U programu GeNML není známá hodnota pro zpracování HUMHDAB. Programy jsou seřazeny podle dosažené průměrné hodnoty sestupně zleva doprava.

Postupem času se schopnost komprese DNA sekvencí zlepšovala. Novější metody stále posunovali hranici dosažených výsledků. Především sofistikované Aritmetické metody (GeNML, XM) dosahují nízkých hodnot. Čím větší počet pravděpodobnostních modelů mají k dispozici, tím efektivnější úrovně komprese je dosaženo.

Ve srovnání s těmito programy zůstává DNAcod se svými dosaženými hodnotami na posledním místě. Nemusí se jednat přímo o neefektivní návrh metody, svoji roli určitě hraje úroveň implementace. Např. programy gzip a zip z předchozího listu – oba pracují na stejném principu, ale dosahují odlišných výsledků.

Závěr

Během této práce jsem se podrobně obeznámil s problematikou komprese DNA a poznal rozdílné metody řešení. Vyzkoušel jsem několik metod alternativního zápisu biologických sekvencí, které ale v důsledku nepokořily kompresní poměr 2bitového kódování.

Nakonec jsem navrhnul postup detekce opakovaných řetězců, který je založen na tříděném seznamu sufixů vstupní sekvence. Tuto metodu a vlastní způsob interpretace výskytů jsem naimplementoval do nového programu DNAcod, ve kterém dále kombinuji slovníkovou a statistickou metodu komprese.

Programem jsem následně zakódoval sadu testovacích DNA sekvencí. Průměrný dosažený kompresní poměr dosahuje hodnoty pod 2 bity na bázi. Tento výsledek předčí hodnoty testovaných univerzálních kompresních algoritmů jako je např. 7zip, avšak nedosahuje výsledků specializovaných programů pro kompresi DNA, které jsou vyvíjeny skupinami expertů z oblasti bioinformatiky.

V algoritmu DNAcod je určitě prostor pro následné vylepšení. Především bych se zaměřil na rozšíření implementace o práci s invertovanými výskyty a slučování nepřesných výskytů. Také nasazení vyšších řádů Aritmetického kódování by mohlo zvýšit výslednou úspěšnost komprese. I přes tyto nevyčerpané možnosti, je hlavním přínosem této práce především provedení rozboru a popisu existujících metod z oblasti komprese DNA.

Použitá literatura

[1] VOET, D. a J. VOET. Biochemistry. 4th ed. Hoboken, NJ: John Wiley, c2011, xxv, 1428, 53 p. ISBN 04-709-1745-8.

[2] ŠÍPEK, A. jr.: Genetika - Biologie; Váš zdroj informací o genetice a biologii [online]. [cit. 2014-01-21] Dostupné z:http://www.genetika-biologie.cz.

[3] DNA. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-01-23]. Dostupné z: http://cs.wikipedia.org/wiki/DNA.

[4] National Human Genom Research Institute (NHGRI) [online]. [cit. 2014-06-24]. Dostupné z: http://www.genome.gov.

[5] STEIN, L. D. The case for cloud computing in genome informatics. Genome Biology [online]. 2010, roč. 2010, č. 11 [cit. 2014-01-21]. DOI: 10.1186/gb-2010-11-5-207. Dostupné z:http://genomebiology.com/2010/11/5/207.

[6] Česká republika se stala členem EMBL. In: Ministerstvo školství, mládeže a tělovýchovy: Tiskové zprávy [online]. [cit. 2014-01-27]. Dostupné z: http://www.msmt.cz/ministerstvo/novinar/ceska-republika-se-stala-clenem-embl.

[7] DNA Sequence formats. Genomatix [online]. [cit. 2014-01-28]. Dostupné z: http://www.genomatix.de/online_help/help/sequence_formats.html.

[8] SALOMON, D. *Data Compression: the complete reference*. 4th ed. London: Springer, 2007, xxv, 1092 s. ISBN 978-1-84628-602-5.

[9] VEČERKA, A. *Komprese dat*. Univerzita Palackého, Přírodovědecká fakulta, Katedra Informatiky. Olomouc. 2004. Dostupné z: http://phoenix.inf.upol.cz/esf/ucebni/komprese.pdf.

[10] GRUMBACH, S. a F. TAHI. Compression of DNA sequences. *DCC*, pages 340–350, 1993.

[11] GRUMBACH, S. a F. TAHI. A new challenge for compression algorithms: Genetic sequences. *Inf. Process. Manage.*, 30(6):875–866, 1994.

[12] RIVALS E., J.-P. DELAHAYE, M. DAUCHET a O. DELGRANGE. A guaranteed compression scheme for repetitive DNA sequences. *DCC*, page 453, 1996.

[13] CHEN, X., S. KWONG a M. LI. A compression algorithm for DNA sequences and its applications in genome comparison. *RECOMB*, page 107, 2000.

[14] LOEWENSTERN, D. a P. N. YIANILOS. Significantly lower entropy estimates for natural DNA sequences. *Computational Biology*, 6(1):125–142, 1999.

[15] MATSUMOTO, T., K. SADAKANE a H. IMAI. Biological sequence compression algorithms. *Genome Informatics*, 11:43–52, 2000.

[16] WILLEMS, F., Y. SHTARKOV a T. TJALKENS. The context-tree weighting method: Basic properties. *IEEE Trans. Info. Theory*, pages 653–664, 1995.

[17] SATO, H., T. YOSHIOKA, A. KONAYAGA a T. TOYODA. DNA Data Compression in the Post Genome Era, *Genome Informatics*, vol. 12, pages 512–514, 2001

[18] CHEN, X., M. LI, B. MA a J. TROMP. DNACompress: Fast and effective DNA sequence compression. *Bioinformatics*, 18(2):1696–1698, 2002.

[19] MA, B., J. TROMP a M. LI. Pattern Hunter: faster and more sensitive homology search, *Bioinformatics,* 18(3), 440–445, 2002.

[20] TABUS, I., G. KORODI a J. RISSANEN. DNA sequence compression using the normalized maximum likelihood model for discrete regression. *DCC,* page 253, 2003.

[21] TABUS, I. a G. KORODI. An efficient normalized maximum likelihood algorithm for DNA sequence compression. *ACM Trans. Inf. Syst.*, 23(1):3–34, 2005.

[22] TABUS, I. a G. KORODI. Genome compression using normalized maximum likelihood models for constrained Markov sources, *IEEE Information Theory Workshop*, Porto, Portugal, May 5-9, 2008.

[23] CHANG, C. DNAC: A Compression Algorithm for DNA Sequences by Nonoverlapping Approximate Repeats. *Master Thesis*, 2004.

[24] BEHZADI, B. a F. LE FESSANT. DNA compression challenge revisited: A dynamic programming approach. *CPM,* pages 190–200, 2005.

[25] CAO, M. D., T. I. Dix, L. Allison a C. Mears. A Simple Statistical Algorithm for Biological Sequence Compression, *DCC*, 43–52, 2007.

[26] ODERO, Agola Kisira. Arithmetic Compression With C#. In: CodeProject [online]. 2009 [cit. 2014-01-05]. Dostupné z: http://www.codeproject.com/Articles/45320/Arithmetic-Compression-With-C

[27] PLATOŠ, Jan. Komprese dat. In: HomeL VŠB CZ [online]. [cit. 2014-01-05]. Dostupné z: http://homel.vsb.cz/~pla06/kod.php

Seznam příloh

Součástí BP/DP je CD/DVD.

Adresářová struktura přiloženého CD/DVD:

Appendix – přílohy

Binary – spustitelný soubor DNAcod.exe (Win32 Console App., .NET Framework 4.5)

Example – dávkové soubory pro ukázku kompresi, dekompresi a porovnání souborů

Source – zdrojový kód (Visual Studio 2013 Solution)

TestingDNA – skupina 9 testovacích DNA sekvencí (Plain format)

TestingDNA\Processed – výsledné soubory po zpracovaní, program a dávka pro hromadné spuštění

Thesis – text této diplomové práce

obsah.txt – popis obsahu

Příloha A: *Porovnání výsledků komprese*

Sekvence	*gzip*	*zip*	*7zip*	*DNAcod*	*BioComp2*	*GenComp*	*CTW-LZ*	*DNAComp*	*DNAPack*	*CDNA*	*GeNML*	*XM*
CHMPXX	2,2821	2,1600	2,0820	1,9525	1,6848	1,6730	1,6690	1,6716	1,6602	-	1,6617	1,6577
CHNTXX	2,3347	2,2288	2,1624	1,9907	1,6172	1,6146	1,6120	1,6127	1,6103	1,6500	1,6101	1,6068
HEHCMV	2,3277	2,2109	2,1642	2,0147	1,8480	1,8470	1,8414	1,8492	1,8346	-	1,8420	1,8426
HUMDYST	2,3627	2,3338	2,2291	1,9537	1,9262	1,9231	1,9175	1,9116	1,9088	1,9300	1,9085	1,9031
HUMHBB	2,2454	2,1893	2,0573	1,9376	1,8800	1,8204	1,8082	1,7897	1,7771	1,7700	-	1,7513
HUMHDAB	2,2395	2,1644	2,0222	1,9884	1,8770	1,8192	1,8218	1,7951	1,7394	1,6700	1,7059	1,6671
HUMHPRTB	2,2667	2,1976	2,0581	1,9757	1,9066	1,8466	1,8433	1,8165	1,7886	1,7200	1,7639	1,7361
MPOMTCG	2,3290	2,2204	2,1276	1,9911	1,9378	1,9058	1,9000	1,8920	1,8932	1,8700	1,8822	1,8768
VACCG	2,2519	2,1297	2,0665	1,9513	1,7614	1,7614	1,7616	1,7580	1,7583	1,8100	1,7644	1,7649
Průměr	*2,2933*	*2,2039*	*2,1077*	*1,9728*	*1,8266*	*1,8012*	*1,7972*	*1,7885*	*1,7745*	*1,7743*	*1,7673*	*1,7563*

Hodnoty kompresního poměru jsou uváděny v jednotkách bpb. Sloupce jsou seřazené sestupně podle průměrného kompresního poměru